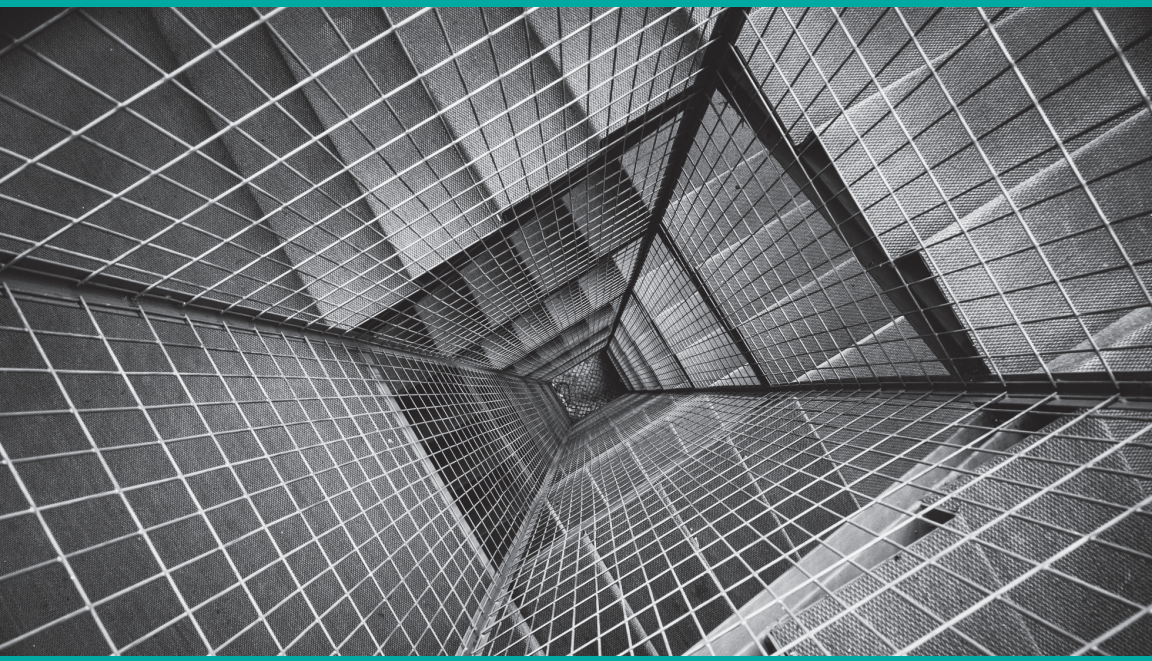
O'REILLY®

Compliments of IMPERVA®

Web Application Firewalls

Securing Modern Web Applications



Chad Russell

Protect Your Applications Wherever They're Deployed

Imperva helps you protect your applications from hackers and unknown vulnerabilities.

Our top-rated web application firewall is available on-prem or in the cloud—or even a combination of both.

Imperva WAF lets you:

- Profile application behavior and apply virtual patches
- Stop OWASP Top 10 attacks and more with crowdsourced threat intel
- Start blocking threats immediately while offering custom rule capability

Find out more about how Imperva can help protect your business.
https://www.imperva.com/products/application-security/

IMPERVA®

Web Application Firewalls

Securing Modern Web Applications

Chad Russell

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Web Application Firewalls
by Chad Russell

Copyright © 2018 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Courtney Allen
Production Editor: Colleen Cole
Copyeditor: Octal Publishing, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

March 2018: First Edition

Revision History for the First Edition

2018-03-12: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Web Application Firewalls*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Imperva. See our *statement of editorial independence*.

978-1-492-03228-1

[LSI]

Table of Contents

Introduction v

1. Current Application Threats and Challenges 1
Code Complexity, Microservices, and Third-Party Libraries 1
Microservices and Container Security 3
Industrialization of Attacks Using Botnets 6
Gaining Access to Data Through Code Manipulation or Sensitive Credential Compromise 8

2. Types of Attacks 11
The OWASP Top 10 12
Business Logic Attacks 18
Predictable User Names 19
Avoid Weak Passwords 19
Model Threats During the Design Phase 20
Distributed Denial of Service Attacks 20
Online Fraud 21
Social Engineering 22
Malware 23

3. Evolution of Firewall and Web Application Firewall Technology . . . 27
Traditional Intrusion Detection System and Intrusion Prevention System Technology 27
Next Generation Firewalls 28
WAF Technology 29
Detecting and Addressing Application Layer Attacks (SQL Injection, Cross-Site Scripting, Session Tampering) 30

Core WAF Capabilities 34
Anatomy of an XSS Attack 36
WAF XSS Filters and Rules 38
How WAFs Can Protect Against Session Attacks 38
Minimizing WAF Performance Impact 38
WAF High-Availability Architecture 40
WAF Management Plane 40
Emergent WAF Capabilities 41
WAFs and Their Part in SOC Modernization 46
WAFs Authentication Capabilities 50
Malware Inspection and Sandboxing 50
Detecting and Addressing WAF/IDS Evasion Techniques 51
Adjacent Solutions and Technologies 53
WAF Deployment Models 61

4. Designing a Comprehensive Network Security Solution..... 65
XYZ Corp 65

Afterword..... 71

Introduction

Web Application Firewalls (WAFs) represent the most advanced firewall capabilities in the industry. Traditionally, firewalls had been focused on network layer traffic, but as attacks became more advanced and climbed up the ladder of the Open Systems Interconnection model, a different kind of inspection was needed. A type of inspection that could not only understand and make sense of network traffic but that could also track session state and ultimately make sense of what was taking place at the application layer.

Arguably, most of the complexity and analysis is needed at the app layer due to the large number of protocols and communication formats that are increasing at a rapid rate. Not only do WAFs need to understand the formats and protocol structures at the application layer, but they need to be able to parse the "good" from the "bad" traffic. WAFs can accomplish this type of protection through several means. One such method is signature-based detection in which a known attack signature has been documented and the WAF parses the traffic looking for a pattern match. Another method involves the application of behavior analysis and profiling. Advanced WAFs can conduct a behavioral baseline to construct a profile and look for deviations relative to that profile.

Throughout this book, we cover topics, including the current application threat landscape, types of attacks, the evolution of WAF technologies, and modern deployment architectures. This report will help you to get you up to speed on the latest developments in the space to better understand how you can incorporate and integrate WAF technology with your existing and planned technology deployments, including cloud, on-premises, and hybrid topologies.

Some years ago, attacks on applications and infrastructure were perpetrated by individual hackers in a manual fashion. In an effort to become more efficient and drive more results, malicious operators and organizations have largely automated and industrialized attacks through the use of distributed botnets.

Applications and the way they are developed have gone through significant changes with the advent of cloud deployments, container technologies and microservices. Developers are always interested in reusing other people's code to the maximum extent possible in order to achieve outcomes and functionality for their respective applications. As such more and more third-party libraries are being used during the application development process than ever before. Attackers are aware of this and are looking to take advantage of vulnerabilities found in commonly used third-party libraries such as OpenSSL, for instance. Essentially, this means that the number of well-known vulnerabilities multiplies exponentially the more they are used in the development process. Many DevOps environments are not yet mature enough to address these vulnerabilities in an automated and repeatable way throughout the application development life cycle. Although it's ideal to address it at the outset, it's not always possible due to constant introduction and discovery of new vulnerabilities in those libraries. WAFs and adjacent technologies can help provide gap protection in the form of signature-based and behavior-based identification and blocking, which can help address not only known vulnerabilities and threats, but zero-day threats and vulnerabilities, as well.

This report covers the Open Web Application Security Project (OWASP) Top 10, which outlines the most prevalent vulnerabilities found in applications, and walks through the means of mitigation by way of compensating controls. You will learn about the specifics of WAF functionality as well as emerging functionality and integrations with adjacent security technologies to help you understand where WAFs fit in your overall technology design.

Adjacent WAF technologies and functionality include the following:

- API gateways
- Bot management and mitigation
- Runtime Application Self-Protection (RASP)
- Distributed Denial of Service (DDoS) protection

- Content Delivery Networks (CDNs)
- Data Loss Prevention (DLP)
- Data Masking and Redaction
- Security Information and Event Management (SIEMs)
- Security orchestration and incident response automation

We will address various deployment models, which take the following into consideration:

- On-premises
- In-line reverse proxy
- Transparent proxy/network bridge
- Out of band/port mirroring/Secure Sockets Layer (SSL) termination
- Cloud
- Multitenancy
- Single tenancy
- Software appliance based
- Native cloud
- Hybrid

In the last chapter, I present several use cases and will work through recommended technologies and deployment models based on a given set of business and technical requirements.

CHAPTER 1

Current Application Threats and Challenges

Code Complexity, Microservices, and Third-Party Libraries

The explosion of open source code has been exponential over the past decade. For developers, this means lots of choices about which libraries to use to minimize development effort. If developers can use a library that manages the ugly underpinnings of encryption, they will. Ultimately developers are tasked with facilitating business outcomes. Any digital plumbing that they can take advantage of in the form of third-party libraries is a boon for productivity.

Savvy attackers are keenly aware of this and are constantly looking for zero-day or even published vulnerabilities that they can exploit in commonly used libraries. OpenSSL is a core example. The OpenSSL library handles core encryption functions, so developers don't need to. It's one of the most commonly used third-party libraries. As a result, any vulnerability discovered in such a common core security library carries serious security ramifications.

The Heartbleed Bug was a serious vulnerability in the OpenSSL cryptographic software library. The vulnerability allowed for the theft of information using popular SSLand Transport Layer Security (TLS) protocols that are used to secure much of the communications on the web.

Initially keeping track of these libraries wasn't that difficult because there was a core set that most developers used. Now, however, it's estimated that there has been such an exponential increase in the number of third-party libraries used for many common development projects that keeping track of all of them from a security and vulnerability standpoint has become a significant challenge. For the moment, this involves in-house developed applications in particular, but this could affect shrink-wrapped applications or cloud services that use these libraries, as well.

One of the newer trends in the development world is the use of *microservices*. Microservices, as the name suggests, involves deploying multiple, small, and discrete services. Microservices allow development teams to deploy new functionality iteratively and in quick, small sprints. This is in contrast to the notion of Waterfall development which was traditionally a "big bang" release model. Each microservice potentially represents its own unique attack surface that can be exploited. Developers can and will incorporate third-party libraries in these respective microservices as needed. Many times, in modern DevOps environments these are separate development teams that operate independently and publish their microservice APIs to others, which allows for a loose-coupling methodology. The security ramifications here are that more individual attack surfaces and vulnerable third-party libraries can be introduced and expose your organization to additional risk.

Let's assume for now that we are just focusing on in-house development. Application development is the realm of the Wild West. Traditionally, anything goes in terms of usage in favor of rushing functional bits of software out the door via Agile development methodologies. Developers have full freedom to pull third-party libraries from anywhere on the web. What if they are downloading and using versions of these libraries that have been modified with backdoors or other malicious code? What if they are downloading and using older versions with known vulnerabilities? Cyber security engineers and Security Operations Center (SOC) analysts wake up in cold sweats over issues like these. They are very real, and they either need to be addressed in the software development life cycle or by way of patching and remediation. Essentially, prevention or detection and correction.

Some good news here is that with the advent of DevOps the ability to lock down source libraries through programmatically managed

pipelines and build processes has greatly increased. But, even then, it's all about design and process. Best practices in DevOps that help address the introduction of security risks via third-party libraries include the following:

1. Compiling the libraries from source code.
2. Ensuring that the source code is pulled from a trusted or authoritative source.
3. Always using the latest versions of third-party libraries.
4. Using DevOps best practices such as constant environmental refresh or "repaving" to rebuild images (operating system images or containers) that are immutable in nature.

Although these are great principles, many development teams are still in the early phases of adopting mature DevOps deployments and in the meantime this needs to be balanced with compensating controls. The compensating controls I'm referencing here involve detection and correction. Detection might be achieved by way of regular vulnerability scanning or virtual patching and attack detection by using Web Application Firewalls (WAFs).

Microservices and Container Security

Figure 1-1 depicts an Amazon Web Services (AWS) Elastic Container Service (ECS) implementation. It's a container orchestration engine similar to Kubernetes in function. Rather than building out your own container engine such as Kubernetes, Amazon provides container orchestration as a service.

In Figure 1-1, you are looking at an unprotected set of microservices running in a single Virtual Private Cloud (VPC). VPCs are virtual clouds/overlay networks. You can see in this example that the VPC spans two availability zones (Availability Zones 1 and 2 in the diagram).

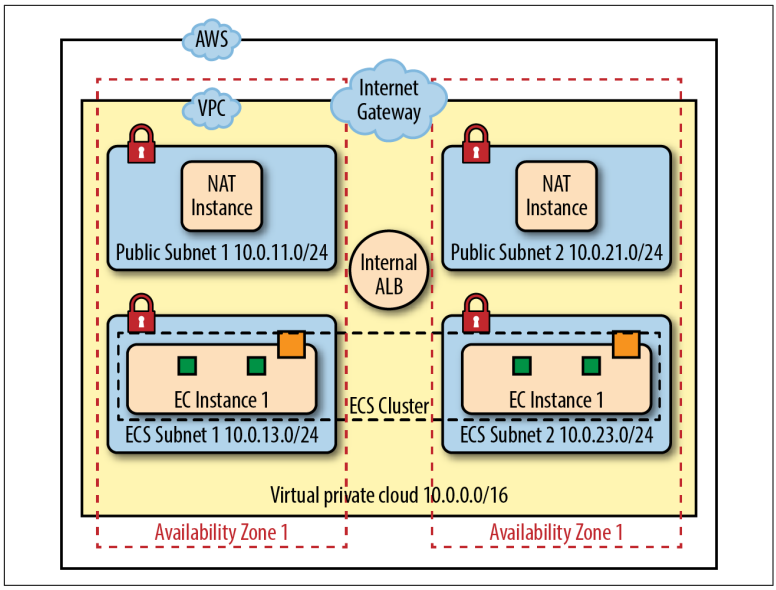


Figure 1-1. An AWS ECS implementation

Figure 1-1 exhibits simple protections in place in the form of IP/Port-based Security Lists. But as we know, this is very rudimentary and does not address security beyond Layer 3.

There are two ECS private subnets that house the Docker instances. There is a single ECS cluster that spans the two private ECS subnets. Each respective ECS instance in the cluster can run multiple docker container images.

Notice that there are two Network Address Translation (NAT) instance subnets, one in each respective availability zone. This is a requirement for Amazon Elastic Compute Cloud (AWS EC2) services. AWS EC2 services are not allowed to have direct access to the internet. Lastly, in this "unprotected" environment there is an internal AWS Load Balancer (ALB) distributing traffic across ECS container instances. In this setup, the Amazon EC2 instances are not directly accessible from the internet.

Now, I'm going to walk you through an example of a WAF-protected AWS EC2 container deployment. First, let's take a look at Figure 1-2.

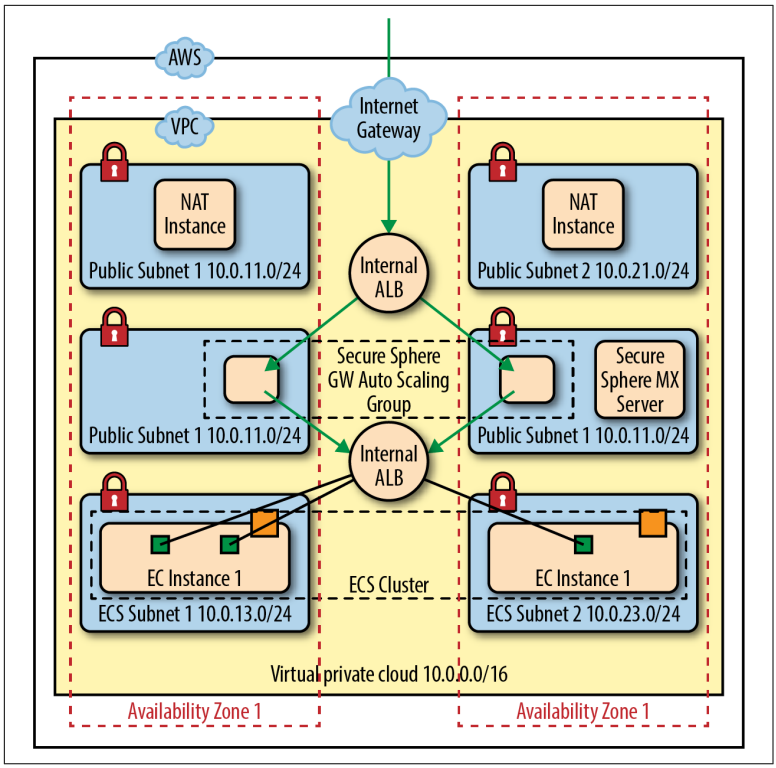


Figure 1-2. A WAF-protected AWS EC2 container deployment

In Figure 1-2 we've added some additional components to make the container-based microservices accessible from the internet. First, we've created two new subnets that will be allocated exclusively to two WAF virtual appliances. Next, we've placed these two virtual WAFs into their respective subnets. We are routing traffic from the WAFs to the internal ALB.

We've also added an external facing ALB, which translates public IPs to the private IP address targets of the internal WAF appliances. Our external DNS will resolve to IP addresses bound to the external interface of the external ALB.

Now that the WAFs have been introduced, all of the respective microservices running in Amazon EC2 Container Services are now protected from OWASP Top 10 attacks, account takeover, and many other threats that we cover in greater detail throughout this book.

It's worth noting that because these WAFs are virtual appliances they can be deployed automatically using AWS CloudFormation templates to automate the deployment process completely.

Note also the presence of a WAF management server. This server is placed in one of the WAF subnets and can be accessed securely for management purposes via a jump-box to limit administrative access.

Any seasoned security professional will tell you that there is no silver bullet. In this case, we are using WAFs to address the "detect" and "correct" aspects of runtime application security that might not have been addressed in the continuous integration (CI)/continuous deployment (CD) pipeline. Having a defense-in-depth strategy can help you to ensure that whatever you don't catch in development doesn't come back to haunt you in production.

Industrialization of Attacks Using Botnets

Automation in and of itself is not necessarily a new tool for attackers. However, there are several trends that have converged that have allowed them to become much more efficient in compromising application data across the web. One of these trends is the use of botnets.

Enterprising hackers are entrepreneurial in nature. This means that they are looking for gains in efficiency, impact, and results. Hackers will take advantage of the newest innovations to better scale their efforts. *Botnets* are usually geographically dispersed groups of machines that have been compromised and back-doored by some sort of malware and are now under the remote control of a centralized group of botnet controllers. Initially, botnets were largely used to facilitate Distributed Denial of Service (DDoS) attacks. This proved effective because in order to block against DDoS attacks the typical methodology is to blacklist individual or groups of IP addresses. If all of the attacks are coming from one identifiable region, the potential impact on your ecommerce sales will likely be low as a result of blocking this range of addresses. DDoS attacks utilizing botnets have become game changers for attackers. The reason is that an attacker can command and control geographically disparate zombies (compromised machines under the control of the botnet commander) which will generate traffic to try to bog down the website to the point that it is inaccessible by legitimate customers. As you can imagine, these zombies can potentially do much more

than simply inundate a website with traffic. Attackers have now industrialized botnets in such a way that they can use these botnets to automate attacks for different purposes.

One outcome for repurposing these botnets might simply be to grow the size and scope of the botnet itself. Think of botnets as groups of finite resources ready to be instructed for a given activity. Many times these "activities" are described as campaigns. Now, wait a minute, usually when you use the word *campaign* you might hear it in the context of an "advertising campaign." Well, as I mentioned earlier, hackers are entrepreneurial in nature, and their campaigns are trying to get the word out, too, but in a more forceful and malicious manner.

Building botnets takes time and resources, so an emerging trend in this space is the notion of botnets-as-a-service. On the dark web, hackers can purchase these services using Bitcoin or Ethereum to do their bidding for a subscription fee. Again, if this sounds like a structured, legitimate business, you are correct, with the exception of the legitimacy of it. This means that attackers can essentially rent botnets to execute their own campaigns.

Another potential use of botnets outside of simply growing their own networks or executing DDoS attacks is to execute surgical strikes against websites properties and applications. Botnet zombies can be orchestrated to achieve complex tasks in tandem with one another.

One regiment within a botnet might be commanded to create a diversion via a DDoS attack. This is a very common tactic as part of a larger orchestrated attack. The purpose of staging a DDoS attack in this case is to create a diversion that might overwhelm Intrusion Detection Systems (IDS's), firewalls, and security logs with noise. Think of this noise as a haystack. The purpose of the attacker is to begin surgically implanting needles into these DDoS haystacks.

As part of this industrialized, botnet-as-a-service orchestrated attack, the attacker might have a separate regiment of the botnet carrying out surgical scans of the target network under the cover of the DDoS attack. All the while the attacker is collecting useful information about the attack surface of the application. Subsequently, another regiment within the botnet can be directed to carry out an exploit against an identified component of the site. For example, suppose that a web server is vulnerable to a new vulnerability in a

third-party library that allows one of the botnet zombies to access the underlying operating system (OS) and is successful in gaining access to a shell and alerts the botnet commander (the human). This is where a real human hacker might take over and begin moving laterally or deeper into the network to exploit other systems by hand.

Theoretically, even the activities within the network could even be further automated by another botnet regiment but it might be noisy. Meaning that a skilled human hacker will likely be more effective at this point.

If this example sounded a lot like a historical account of the D-Day landing in World War II, you are beginning to get the picture. A well-orchestrated and industrialized system with a defined command and control substrate. This is the industrialization of cyberattacks through botnets.

Gaining Access to Data Through Code Manipulation or Sensitive Credential Compromise

It's estimated that 50% of cyberattacks involve compromised credentials. The system of using usernames and passwords to gain access to websites is one that is fundamentally broken but this paradigm continues to perpetuate. The issue, of course, is that fact that users reuse the same username and password information for multiple sites. Therefore, when one of these sites is hacked, bad actors can reuse the credentials to attack other sites.

For attackers, using compromised credentials is the simplest way in the front door. Hackers don't care how elaborate an attack is, they are interested in the end result. They want to expend the least amount of effort.

You can categorize account compromise into two key buckets. Let's take a look at those.

End-User Accounts

The first bucket is end-user accounts. This is in line with the aforementioned description about compromised accounts end-user accounts. So, when Yahoo! is hacked, bots can use those credentials by way of credential stuffing attacks. Basically, the botnet is config-

ured so that the variables of username and password are replaced with compromised username and password data in succession by the botnets. These repositories of hacked usernames and passwords can be found on the dark web for sale to anyone who wants to pay the Bitcoin going-market value. And they are not just being sold to one buyer; they are resold over and over again to anyone that wants to pay pennies on the dollar. This of course isn't limited to usernames: it also can include social security numbers, credit card information, and so on.

Sensitive and Privileged Accounts

Another category of account compromise is sensitive or privileged accounts. These are accounts that have administrative privileges over the OS, databases, and network devices. Let's revisit the "industrialized attack-bot" example from the previous section. The botnet had gained a foothold to an internal system and had shell access. At this point, the remainder of the attack was to be handed off to a skilled human hacker. Now the account that the hacker has gained access to is not a privileged account. It's incumbent upon the hacker to escalate his privileges. The attacker now has some advantages in his favor by way of having local access to the system. If you've ever read through a vulnerability report, you've probably noticed that they are typically classified by attack vectors such as "local" or "remote." Network and system administrators typically focus on patching remotely accessible vulnerabilities before local vulnerabilities in terms of priority. A likely next step for our hacker is to enumerate as much information as possible about the system to which he has shell access. Useful information will be OS, version, username you are logged in with, and version information about any other software packages running on the system. Armed with this information, an attacker can reconcile vulnerable software versions and identify potential locally exploitable vulnerabilities that allow for escalation of privilege. After a vulnerable library or package has been identified, an attacker can research the web for known exploits. Attackers can gather this type of information from sites such as Packet Storm and CVE Details, among others.

An attacker has many potential avenues to explore, and the point here is not to identify all the permutations but to illustrate the process. Suppose that the attacker is in a Linux shell. The attacker might look for SUID (Set User ID) and SGID (Set Group ID) vulnerabili-

ties. These are the bits that are set when a given application or binary needs to execute with the privileges of a particular user or group. Without getting into too much detail, the attacker can potentially exploit the fact that these applications need to run with escalated privileges. The attacker can thus use this to escalate their own privileges to those of "root," thereby escalating their privileges. Now, that attacker has effectively gained full control of the system to which they have shell access. They can now fully control this machine to gain lateral or forward access into the network. This can be used as a launchpad to further compromise other systems and data throughout the environment.

There are many other ways by which attackers can gain access to sensitive credentials. I just walked through an example of escalating one user's privilege to that of another user. Remember that attackers are looking for the easiest way to achieve their objectives, and they have numerous means by which they can actually steal sensitive credentials as opposed to going through the process of privilege escalation.

Attackers can use botnet-driven social engineering methods, including those that deliver malicious payloads such as key-loggers to harvest sensitive credentials. All an attacker needs to do is fool a user who has privileged access into giving up their username and password information. That might sound difficult, but it really isn't. If an attacker can harvest the email addresses of database and system administrators of Company X from a publicly accessible source such as LinkedIn, they can program their botnets to perform automated spearphishing campaigns. These emails might masquerade as messages from the Help desk team that require the administrator to log in to a portal page and change their password. All an attackers needs is for one administrator to fall for it.

In this chapter, we covered current application threats and challenges, such as the use of third-party code, the industrialization of botnets, and how sensitive credentials can introduce vulnerabilities into your compute environment. In Chapter 2, I break down the types of attacks in detail using various classifications to help you better understand the details of the current threat landscape.

CHAPTER 2
Types of Attacks

In this chapter, I classify attacks by using the Open Web Application Security Project's (OWASP) Top 10 list, discuss business logic attacks, and cover Denial of Service (DoS) attacks. The OWASP Top 10 is the industry standard taxonomy for categorizing application-level vulnerabilities and attacks. In the business logic section, you will learn how poor coding logic can contribute to risk exposure. Finally, we wrap up this chapter with a discussion specific to DoS attacks.

Application-layer attacks come in many forms. OWASP has become the definitive source for tracking and trending application-layer security vulnerabilities. The purpose behind the open source organization is to improve the security of software and to ensure that individuals and organizations are educated about software security in order to empower them to make informed decisions regarding security.

Application-layer attacks and vulnerabilities are inherently more complex than most network-layer attacks because they involve the inner workings of applications, which differ greatly from one application to the next.

There are really two ways of looking at application-layer vulnerabilities. The first is from the perspective of the developer. The developer's perspective is all about prevention and testing before release. The application of development best practices such as input validation help to head off vulnerabilities from the outset.

The other perspective is that of the organization deploying, hosting, or consuming the application. If your organization deploys software that it did not develop, your first line of defense will be patching. Having a proactive software vulnerability assessment and patch management program afford operations teams the opportunity to detect and correct vulnerabilities in their environments.

Categorically the OWASP Top 10 addresses vulnerabilities specific to misconfiguration, injection and improper access controls.

The OWASP Top 10

The OWASP Top 10 has evolved since it's inception in the early 2000s. In the 2017 iteration of the list, there are a few new additions. Some of these tend to be a function of the cloud computing paradigm.

Here is the iteration of the OWASP Top 10 as of 2017:

- A1: Injection
- A2: Broken Authentication
- A3: Sensitive Data Exposure
- A4: XML External Entities (XXE) (*new*)
- A5: Broken Access Control
- A6: Security Misconfiguration
- A7: Cross-Site Scripting (XSS)
- A8: Insecure Deserialization (*new*)
- A9: Using Components with Known Vulnerabilities
- A10: Insufficient Logging and Monitoring (*new*)

Before we dive into the Top 10 individually, it's important to line out some important terminology. In the descriptions and accounts of the Top 10, you will see particular terms used throughout. Here is a quick rundown of those terms, followed by a brief description of each:

Threat agents
: These are essentially the actors. At some point there is a human in control of orchestrating the delivery of an exploit. The threat agent sometimes is also an extension of a threat actor. Examples

of threat agents acting on behalf of a threat actor include botnets and targeted exploits.

Attack vector
: An attack vector generally refers to the path that a threat actor uses to deliver an exploit. For instance if an attacker constructed spearphishing attacks, the vector could be considered to be a social engineering vector type using email as the attack modality.

Security weaknesses
: These essentially represent vulnerabilities in the target. An example of a security weakness might be having unnecessary ports open on your firewall allowing inbound access to an application on nonsanctioned ports. Another example could be one or more vulnerabilities in a particular version of an application or its underlying platform (database, OS, etc.).

Security controls
: We can consider security controls to be compensating controls. It could be as simple as a firewall rule, or it could be a patch management strategy or your Security Information and Event Management Strategy (SIEM).

Technical and business impact
: This is essentially the potential outcome of an attack. Technical impact might be that the mail servers are knocked offline and the subsequent business impact might be several hundred thousand dollars in lost revenue for the company.

With this brief review of common terms under your belt, let's go ahead and dive into the Top 10.

A1: Injection

Injection comes in several forms. Fundamentally injection involves inserting information that can be used to break out of the intended context of the input. Common categories of injection include Structured Query Language (SQL), NoSQL (Not only Structured Query Language), Lightweight Directory Access Protocol (LDAP), and operating system (OS) command injection.

Injection is interesting and dangerous because it allows an attacker to potentially bypass all existing network, authentication, and

authorization controls in place that protect your application. Injection can sometimes lead to data compromise or even system takeover.

A2: Broken Authentication

Broken authentication involves attack vectors such as stolen credentials, brute-force attacks, dictionary attacks, and session management attacks. As mentioned in Chapter 1, if one website is hacked and a user's password is compromised there, an attacker can use that information as part of a credential stuffing attack on different sites via password reuse.

Brute-force and dictionary attacks involve repeated attempts at authentication (usually automated via botnets) using passwords from a dictionary list or via brute force. Common compensating controls include the use of Captchas, account lockout after multiple failed attempts, and enforcement of password complexity rules.

Compromising session information is a different vector altogether. This might involve the execution of a Man-in-the-Middle attack to capture session data and replay that information as part of a replay attack. In some cases, session IDs are easily predictable.

Compensating controls involve the use of less-predictable session identifiers and the use of digital certificates. Digital certificates help to mitigate Man-in-the-Middle attacks through encryption and through browser notifications indicating a spoofed or untrusted certificate. This is one of the reasons why many websites have mandated an *all HTTPS* strategy when serving content.

The technical and business impact of broken authentication can include data compromise, data leakage, or complete system compromise if the account is a privileged system account.

A3: Sensitive Data Exposure

Sensitive data exposure vulnerabilities are a result of poor data protection practices. Sensitive data can be exposed at rest and in transit. Data that is not encrypted at rest (on a drive or tape) is a prime example. Sometimes, this might involve data backups that are not encrypted and the backups fall into the wrong hands. Standard defense-in-depth strategies typically involve the use of encryption depending upon the sensitivity of the data at hand. For instance, if it

is Personally Identifiable Information (PII) or PCI (credit card data), you should deploy additional protections such as encryption.

Encryption is effective only if you properly manage the keys used to encrypt the data. This means that you need to implement effective key management processes and technologies. From a separation-of-duties perspective, the team that is responsible for managing the keys should not be directly involved with the operational management of the system itself. By separating these functions, it forces some level of collusion to take place in order to compromise the key procedurally.

More modern solutions for key management involve the storage of keys in separate protected *key vaults*, which allow for only indirect access and usage of the key. These key vaults should be managed by a group of security administrators who are not directly involved with the day-to-day administration of the system (a database is a great example).

A4: XML External Entities (XXE) (*New*)

XXE attacks exploit vulnerabilities in XML processing engines. You can consider it to be a form of injection attack. If an unexpected XML entity such as

```
<!UNEXPECTED XXE SYSTEM file:///etc/passwd">]>
```

is passed to the system and proper validation is not in place, that data can be processed in such a way that allows an attacker to break out of the context of the XML processor.

Legacy Simple Object Access Protocol (SOAP) web services prior to v1.2 are often susceptible to XXE attacks. OWASP recommends the use of less-complex data formats such as JSON. If you absolutely need to use XML processing, you should update the XML processors and libraries to the latest versions. Whitelisting of valid inputs can help to ensure that unexpected inputs are not processed.

A5: Broken Access Control

Access controls imply authorization. In some cases, attackers can bypass application authorization mechanisms via URL manipulation, page manipulation, or custom API attacks.

Attackers should not be able to access resources by simply guessing URL strings and patterns. Security through obscurity is not a viable protection pattern. All permutations of URL strings should be protected. Best practices can include the use of *deny-all* patterns and token invalidation after logout. A deny-all pattern for a firewall for instance might start with a rule statement that denies all Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) traffic with subsequent rules that open specific ports such as port 80 for HTTP.

A simple example of this type of attack might involve the manipulation of a URL such as *https://mywebsite.com/products*.

Changing this URL to *https://mywebsite.com/products?purchasedby=user@email.com* might allow a user to directly access resources not explicitly authorized. For an attacker, this might require only simple trial and error. An attacker can use fuzzing techniques to discover unidentified patterns.

A6: Security Misconfiguration

If your house is equipped with the latest alarm systems and locks but none of them are enabled, you could say they haven't been configured properly. The same holds true with software security controls.

Common attack vectors include the exploitation of known administrative accounts and default passwords, unnecessary services, and unpatched systems.

A7: Cross-Site Scripting (XSS)

The OWASP Top 10 Report for 2017 states that "XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications."

XSS is another type of vulnerability that is associated with unvalidated inputs. XSS can affect APIs or web applications directly. I would describe it as a type of injection. The most benign example is the simple JavaScript Alert Dialog box. Most commonly, XSS involves the injection of unauthorized JavaScript commands, but attackers can inject other client-side input such as ActiveX, Java, VBScript, or Flash, as well.

XSS is organized into the following three categories:

Reflected XSS
: Reflected XSS attacks are transient in nature and not persisted. Reflected and Persistent XSS attacks incorporate client injection of code that is subsequently processed by server-side instructions that render manipulated content back to the browser. Reflected XSS attacks are the most prevalent type of XSS attack on the web. HTML query parameters are particular vulnerable to reflected attacks.

Persistent XSS
: Persistent XSS attacks inject code such that it is persisted and written to disk. An example might be an injection into a blog post that writes the injected client-side code into an application's database. When the next user visits the site and renders the blog post from the attacker, the injected client-side script is executed by that user's browser.

DOM-based XSS attacks
: The DOM is the browser's Document Object Model. DOM-based XSS attacks are fully client-side based attacks. The injection manipulates a DOM object and is fully reflected back to the client without any server-side processing.

A8: Insecure Deserialization (*New*)

In practice, serialization involves taking data structures and sequencing them into consecutive bytes of data that can be stored in memory or on disk. Deserialization takes the serialized data and reassembles it back into its original data structure. Examples of data structures that are commonly serialized and deserialized include JSON, XML, HTTP cookies, HTML form parameters, API authentication tokens, and Remote Procedure Calls (RPC) communications.

If an attacker can influence the way that data is deserialized, they can potentially manipulate the reconstituted data structure in a manner that compromises the integrity of the application. To prevent deserialization attacks, it is best to not accept serialized objects from untrusted sources.

A9: Using Components with Known Vulnerabilities

This one is fairly self-explanatory. When application developers use code libraries with known vulnerabilities, they are making the application directly vulnerable. Given the exponential increase in the amount of third-party code libraries that developers use, this becomes a nagging issue in development and production environments.

A best practice is to continually "repave" or redeploy an application's microservices on a frequent basis, which incorporate the latest, patched versions of affected libraries. This works great in mature DevOps environments, but many shops have not reached this level of maturity.

At a minimum the security operations team should be scanning production applications for vulnerabilities and patching the application on a proactive basis.

A10: Insufficient Logging and Monitoring

Even though logging is a detective control in nature, it's absence leads to a lack of visibility as it relates to threats. Applications should be sufficiently instrumented so that security-related events are captured and logged as needed. This allows security operations teams to monitor and correlate this information with other security and network events to facilitate proper threat identification and incident response procedures.

Business Logic Attacks

We cannot address all vulnerabilities with a simple software patch. Some software vulnerabilities are a result of core design flaws in an application. It is exponentially simpler to address design flaws in the development phase rather than trying to reengineer an application after it's been released.

Design flaws are fundamentally different than bugs. Bugs are very local in nature and usually represent some sort of flaw in implementation. Examples might include not sanitizing input and then being subject to SQL injection in a web form. Design flaws represent a miscalculation in requirements, missing requirements, or flawed architecture. Circumventing navigation of a website is a simple example.

Example

If http://site.com *uses* http://site.com/initstep=1 *before an authentication event takes place, maybe we can deduce that changing the value to* http://site.com/initstep=2 *will move us into the next state of the transaction which might actually be a post-authentication step.*

Another typical design flaw is when administrator privileges are not well protected. Maybe the default username and password is still in place. A real-world example of this is illustrated with many home routers and access points. Many of these devices ship with default usernames and passwords so an attacker could then log in to the router and assume administrative privileges.

Abuse of password recovery features is also a common attack vector that stems from flawed design. For instance, if the password recovery questions have to do with what high school I went to then that is something that an attacker could easily research on Facebook. This is flawed design.

Predictable User Names

Using predictable identifiers can also be an issue. Suppose that you know that all user identifiers are an email address. Well, if the company is company.com and you know that the typical user email format is `firstname.lastname`, you already know the user identifier. An attacker can use that information to begin performing a dictionary attack on an account or even manipulate a URI by injecting that user's ID in hopes that it might yield sensitive information.

Avoid Weak Passwords

Weak password policy design allows for users to recycle versions of their passwords using predictable iterations. For instance, a user's password might have been `soccer`, and then the user changes it to `soccer1`. There are obviously more complex examples, but they can be reverse engineered based on the password policy that is defined for all to see when creating the password to begin with. Suppose that a site changes its password policy to require at least one special character. You can make some educated guesses as to how attackers can abuse this.

Address Security in the Design Phase

So, how can software developers address these issues earlier in the design phase? Well, in agile development shops developers use the notion of *user stories*. User stories are essentially tidbits of functionality that are built in to a website or microservice. Developers can incorporate *security stories* in their development processes as well as in the design phase and add these stories to their development backlog.

Model Threats During the Design Phase

Another best practice is to engage information security teams early in the software conception and design phase. This allows security teams an opportunity to do threat modeling against user stories and proposed software architectures. Out of these threat modeling sessions new security features or security stories can be identified and incorporated into the development backlog as deemed appropriate. I've seen this trend manifest itself, especially in DevOps environments. Threat modeling is now trending toward "the left," meaning toward the software development phase as opposed to an exercise that takes place right before a production release.

There are challenges associated with threat modeling directly before release. Suppose that the security identifies a fundamental design flaw. What is realistically going to happen to the go-live date? Well, I can tell you that in the real world the application is usually released anyway and the business requests that the security team identify some compensating controls that can be put in place until the components in question can be reengineered. In practice, this leads to a situation in which the application will be vulnerable for a longer period of time, and, in the meantime, many more hours of labor will be expended reworking the software design rather than if the issue was caught during initial user-story analysis at the outset.

Distributed Denial of Service Attacks

Although I covered a Distributed Denial of Service (DDoS) scenario in Chapter 1, there is a new DDoS trend emerging. There will be more than eight billion devices connected to the internet by the year 2020. At the beginning of the internet boom, primarily connected devices were PCs or laptops. With the emergence of smartphones

and home gaming consoles the average device-to-human ratio tripled.

Queue the Internet of Things

The Internet of Things (IoT) can include any device that has sensors and is network accessible. This can include traffic lights, cameras, security systems, alarms, cars, trains, industrial devices, airplanes, drones—you name it. There is great benefit to users and operators in enabling these devices to be internet connected. They can be monitored, becoming sources of big data for companies to act proactively on behalf of users. For example, even farm tractors are being outfitted with sensors so that manufacturers (such as John Deere) can proactively sense whether a part is about to fail. If the sensors capture this information, the company can proactively send out a replacement part to the tractor operator before the failure occurs.

With all these associated benefits, it's no wonder that these devices are multiplying exponentially across the web. Where there was a two- to three-fold increase in device-to-user ratios with PCs and mobile devices, the IoTs might represent a base of devices that are an exponentially larger presence on the web than users and their personal devices.

Black-hat hackers (hackers with malicious intent) are entrepreneurs at heart. They follow megatrends just like legitimate operators do. IoT devices represent a significant series of opportunities. Perhaps the most significant is the opportunity to take control of these devices and exponentially grow the sizes of their botnets to conduct industrialized DDoS botnet strikes from multiple geographies across the world. These devices can also be sources of critical information in and of themselves, such as industrial control systems.

Online Fraud

Online fraud certainly isn't new, but it is rapidly evolving. Roughly 90 billion ecommerce transactions were tendered in 2016. This is an enormous attack surface for hackers.

Attackers are looking to capitalize on machine learning and artificial intelligence (AI) capabilities to automatically adapt and communicate with victims.

Social Engineering

Social engineering is as old as mankind itself. It's a web of lies, deceit, and treachery. The oldest example of social engineering is a simple lie told from one person to another in order to get someone to divulge information or resources that you want. So what's changed in recent history? Technology and communications have changed. Let's fast forward to 2017–2018 and look at the chatbot trend. Chatbots are being utilized more often to automate communications via SMS and messaging platforms. The most common scenario is automated customer service. But what if an attacker could use AI to interact with a human and socially engineer them in real time? No latent emails from a Nigerian prince are required to execute this tactic. This allows for escalation in real-time fraud and deceit unlike anything we've seen to date.

Of course, the older methods of social engineering are still alive and well. This includes the usual suspects such as phishing, spearphishing, vishing, SMShing, and the like. (Just to clarify, vishing is voice-phishing and SMShing is SMS-phishing.) Notice the only thing that's really changing is the attack and communication vector such as SMS, messaging, voice and email. When in doubt, attackers follow right on the heels of digital marketing trends. They want humans to respond and do their bidding somewhat like legitimate advertisers do. The only difference is in the ethics and intent.

Call centers and Help desks continue to be prime targets for attackers. If a call center can be compromised, corporate accounts and access can be compromised as a result. Many times, when penetration testers are contracted to ethically hack your company, they will request to ethically hack and socially engineer that company's Help desk.

Social media is another vector that has gained popularity and traction with hackers. It's yet another avenue from which they can harvest data and interact with would-be victims to advance their agenda of fraud and illicit profit. Sites such as LinkedIn are a great source of information for spear-phishers. An attacker might search for people with the title of database administrator for a given company. That attacker can subsequently gather social profile information from Facebook and other sources to execute surgical social engineering strikes on the database administrator in an attempt to

gather sensitive credentials such as database administrative accounts.

Malware

One of the main headlines around malware is automation, particularly automation of malware distribution through botnets. Malware such as viruses and worms still carry native capabilities to spread on their own, but with the help of huge industrialized botnets, attackers have increased abilities to deliver malicious payloads.

Although not new, ransomware has been on the rise since 2010. Ransomware signaled a significant shift in malware behavior. Instead of just functioning as adware or stealing computer cycles, ransomware is designed with the intent of making an actual change to the end system.

In this case, the change is to encrypt the data for the purpose of extorting funds from the affected user or organization. So what is the next logical step in the evolution of malware? A worrisome threat is the ability for malware to surgically change data to different values altogether. Let's think about this in terms of IoT trends. Remember all those devices that I referenced earlier such as stoplights, industrial control systems, airplanes, and the like? What if an attacker changed a stoplight at a major intersection from red to green on demand? What if an attacker could gain access to a hospital's life support systems and change the amount of medication in a patient's IV drip? What if a malicious actor could hack into someone's pacemaker wirelessly and induce a heart attack? How about disabling your car's brakes while traveling down the freeway?

Even though IoT shows great promise in advancing humanity, it also represents a significant attack vector. Common means of hacking IoT devices include the use of default credentials and using common shared vulnerabilities among devices. If you want to get a feel for some of the threats I'm calling out about IoT devices, I recommend you visit a site called shodan.io. Think of Shodan as a search engine for the IoT. You can do searches for webcams, traffic lights, refrigerators, and even power plants. It can show you which devices are online and whether they are accessible along with what credentials to use to access some of these devices. You don't even need to be a script kiddie to take advantage of this information. You can use

Shodan to find MongoDB and SQL databases that are openly accessible on the web.

To give you a concrete example, while writing this passage, I went to Shodan and chose the Explore Site option. There is a subcategory called default passwords. When I clicked it, a list of IP addresses opened, along with associated devices, country of origin, and username and password information to access the device. Shodan served up 28,514 results. I could spend the next six months accessing and pillaging data from these devices alone if I had nefarious ends in mind. The types of devices in the list included routers, switches, Cisco devices, and cable modems (very common) from countries such as Egypt, Vietnam, the United States, India, and Brazil. And that was just on the first page.

Now, some of these devices are easily accessible with provided username and password information. It literally doesn't get any easier. And for device listings that don't serve up username and password information, you can always navigate to Exploit Database to correlate exploits with devices and device versions that came up in your Shodan query.

Let's apply some industrialized botnets to the equation. With open and accessible databases of vulnerable systems (Shodan) and exploits (exploit-db) these botnets can dynamically gather this information and execute their own attacks.

Now imagine that these botnets communicate with their command and control (C&C) network to request instructions. These botnet C&C controllers can efficiently dispatch attack workloads across swarms of botnets on demand. This is the malicious side of cloud computing and the IoT.

With all these advances in malware orchestration and capabilities and swarm size, defending cloud and on-premises resources against these threats requires increased processing power, bandwidth, and generates enough alerts to bury most Security Operations Centers (SOCs) the moment they connect their datacenter to the web.

Up until this point, I've spent a good deal of time explaining the cyber security attack and vulnerability landscape. If you are beginning to feel like the odds are stacked against you, you are not alone. The good news is that all of the megatrends that the attackers are using to their benefit can be utilized by the defenders, as well.

Advances in cloud computing, IoT, AI, and Machine Learning are enabling defenders to more effectively fight the onslaught of automated attacks against them.

In this chapter, you learned how to categorize attacks by OWASP Top 10, social engineering, and DoS attacks. Chapter 3 focuses on the use of WAF Technologies. You will learn about the evolution of the technology and how WAFs are being deployed to address many of the common security challenges we covered throughout this chapter.

CHAPTER 3

Evolution of Firewall and Web Application Firewall Technology

So far, we've spent quite a bit of time discussing the modern security challenges and threats. At this point in the book, we are going to shift our focus to solutioning. Before we dive in to the finer points of designing modern solutions using Web Application Firewall (WAF) and adjacent technologies, it's a helpful exercise to walk through the evolution of WAF technology to try to help you understand how and why we arrived at this point in the evolution of WAF technology.

Traditional Intrusion Detection System and Intrusion Prevention System Technology

Let's begin by looking at Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) technology. IDS/IPS technology was the security industries first foray into intelligent parsing of network traffic. IDS/IPS solutions have been traditionally focused on parsing network-level traffic and conducting signature-level comparisons. Network firewalls and IPS systems generally don't provide adequate protection for internet-facing websites on their own. They are generally deployed as adjacent technologies as part of a *defense-in-depth* architecture. Defense-in-depth architecture involves the deployment of multiple layers of protection so that if one layer fails there are other layers to serve as fail-safes. A common example is

the architecture of a castle. There is a moat, a drawbridge, upper defenses, and defenses within the castle walls.

WAFs are generally more adept at terminating Secure Sockets Layer (SSL) traffic and conduct deep application layer inspection at Layer 7. One of the prime benefits of IDS/IPS technologies is their ability to inspect multiprotocol traffic such as FTP, SSH, Telnet, DNS, and other network traffic outside of just HTTP.

IDS/IPS Evasion Techniques

IDS/IPS technologies are not infallible. There are well-documented tools and strategies that bad actors can use to evade detection by these devices. IDS/IPS relies mostly on traffic signatures. Attackers can easily bypass these signatures by modifying parameters within their attack to make their attack look different from the signature. A prime example of this is through the use of *packet fragmentation*.

Suppose that an attack signature showed a particular HTTP payload that constituted a well-known SQL injection pattern. That signature is effective only if the entire SQL injection message is embedded in the same packet fragment. But what if the attacker uses a tool such as fragroute to split up the packet into multiple smaller packets that will ultimately be reassembled by the receiving device? By doing this the attacker has rendered the attack signature ineffective and the attacker has successfully evaded detection and blockage by the customer's IPS/IDS device.

Next Generation Firewalls

In addition to IDS/IPS technologies, there is what's called Next-Generation Firewall (NGFW) technology. Basically, these are appliances that are network firewalls with added IPS/IDS capabilities that can communicate with an authentication store to authenticate users. Sometimes these solutions are described as being *application aware*, which has generated some confusion for consumers. Generally, NGFW devices are deployed to control outbound access to applications to which a user has access as well as how much bandwidth is allocated by application. Many times, NGFW devices are deployed as *forward-proxies*, meaning that they inspect outbound traffic, require users to authenticate, and allow external application access based on a user's Lightweight Directory Access Protocol (LDAP) roles or privileges. But as you can see these devices function more as

outbound content filters and are not designed to provide application-layer protection for inbound traffic applications themselves.

WAF Technology

This brings us to WAF technology. WAFs generally are deployed in cloud or corporate demilitarized zones (DMZs). They can perform SSL termination in order to conduct deep inspection of application layer traffic at Layer 7. They have advanced detection capabilities that allow for protection against zero-day attacks. WAFs are adept at blocking Open Web Application Security Project (OWASP) Top 10 and CWE/SANS Top 25 type vulnerabilities and attacks. Modern WAFs incorporate web attack signatures, web vulnerability signatures, and can use vulnerability scan results. WAFs can provide URL, parameter, cookie, and form protection for applications.

WAFs are deployed inline in-band in a bridge or proxy configuration. They can also be deployed out of band by utilizing port mirroring. WAFs do not just mechanically parse traffic and look for matching signatures. WAFs can analyze application behaviors and establish baselines of acceptable behavior and look for deviations in these communications. This type of capability can allow WAFs to catch zero-day attacks. WAFs can also manipulate inbound and outbound application-layer traffic, which allows for sophisticated capabilities such as *virtual patching*.

In contrast, IPS technology has a minimal understanding of the application layer. It is not designed to protect URLs or their parameters. IPS is not able to determine whether an attacker is scraping your site for sensitive data such as credit card or social information.

WAFs and Virtual Patching

Virtual patches work by analyzing transactions to prevent malicious traffic from making it to the application. These virtual patches essentially prevent the exploit from taking place without having to modify the application's underlying source code. So why not just patch the application, you ask? Well, in many organizations, patches are rolled out at 30-day or longer intervals, thus leaving a system vulnerable between the time the vulnerability is announced and the time that the application is actually patched. Virtual patching is certainly no substitute for actual patching. Actual patching will address

the vulnerability directly. Virtual patches use complex rulesets as opposed to simplistic black-and-white signatures.

Detecting and Addressing Application Layer Attacks (SQL Injection, Cross-Site Scripting, Session Tampering)

WAFs are firewalls that are purposed for protecting HTTP applications. This happens by way of HTTP conversation and content analysis. WAFs incorporate rulesets to thoroughly examine these communications heuristically.

Detecting SQL Injection Attacks

SQL injection attacks have consistently been part of the OWASP Top 10 since the list was originally released. SQL Injection attacks are extremely dangerous because they render all of your defense-in-depth controls useless in one fell swoop. You know that expensive DMZ with dual network firewalls you deployed along with all the operating system (OS) and database access controls you configured? Bypassed.

How does this happen? Well, SQL injection is a type of attack that involves injecting SQL code into a web servers input stream. These input vectors could be forms or even APIs. Let's look at a form-based example. SQL injection and Cross-Site Scripting (XSS) attacks continue to make up the bulk of application-layer attacks on the web today.

Figure 3-1 shows a screenshot of the Mutillidae II vulnerable web application. This is a free distribution you can download to explore examples of how various injection attacks actually work against a virtual machine (VM) instance.

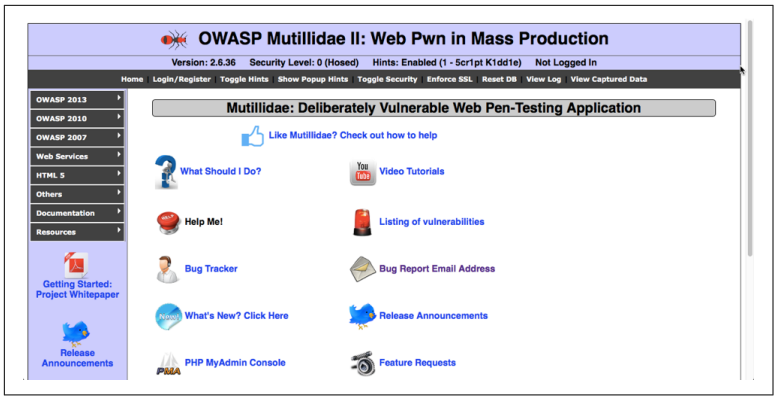


Figure 3-1. The 'Mutillidae II' vulnerable web application

In Figure 3-2, we navigate to a form that we are going to try and compromise via SQL injection.

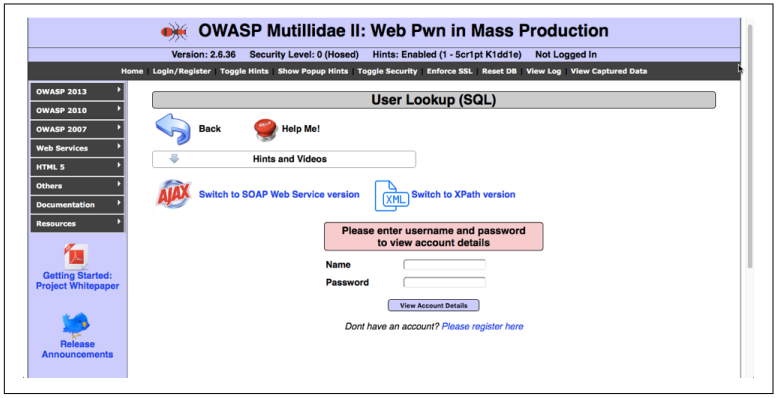


Figure 3-2. A form vulnerable to SQL injection

Before we execute the SQL injection attack, let's talk more about how it works.

All forms usually interact with databases at some point (sometimes LDAP, as well). This application is designed such that the login form is connected to a MySQL database.

SQL injection attacks manipulate data input so that small pieces of SQL instructions are passed back to the database directly. Normally the application takes this form input and constructs a SQL query utilizing "normal form input." Thus, if a form takes username and

password, the SQL query might pull the information and compare it, perhaps by using a SELECT statement.

Some of the most common techniques are to use characters reserved for databases in order to induce error messages. An attacker can use this information to keep prodding the database for information and step through the formulation of a working SQL injection attempt.

Here are some common characters used in SQL injection attacks:

- Single quote
- Backslash
- Double hyphen
- Forward slash
- Period

Let's now tamper with the form input to try to inject a SQL code fragment. Figure 3-3 illustrates what this looks like.

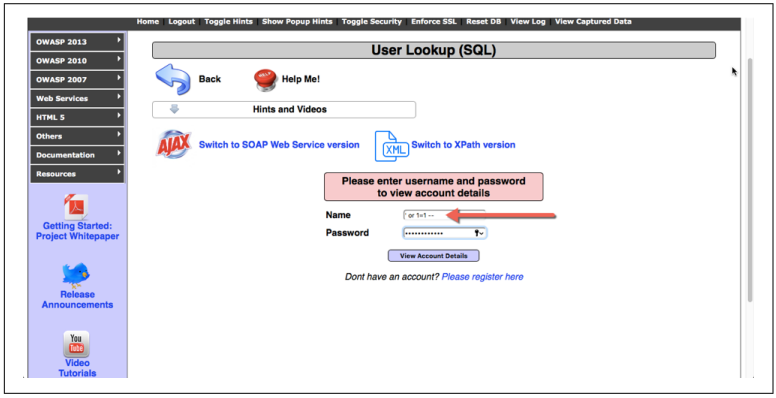


Figure 3-3. Entering a SQL code fragment into the form input

Notice that we are adding a single quote and conditional value that would always evaluate to true. Also notice that we're providing random data into the password field to prevent the form validation from complaining about no data being entered.

Figure 3-4 depicts what you see when you click the View Account Details button.

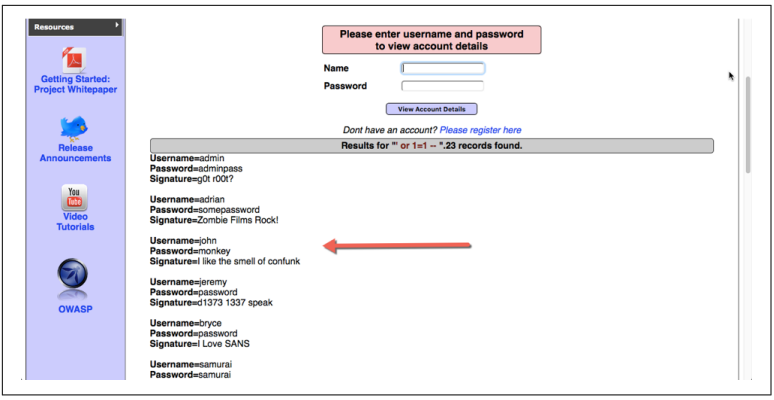


Figure 3-4. A successful SQL injection. The attacker retrieved usernames and passwords stored in the database.

Also notice that we were rewarded with a list of all usernames and passwords in the database.

Now you've had a chance to see how a SQL injection attack works in practice. This is important to understand how WAFs actually protect against these attacks. In this example, what might a WAF be looking for? Well, it's safe to say HTTP POSTS are a key attack vector. WAFs can use rulesets to analyze this input and look for characteristics of SQL injection attacks such as the entry of reserved characters into the form. But there's much more analysis going on under the hood than just a simple scan for reserved SQL characters.

WAF rulesets look for *tautologies*. Tautologies are conditions that always evaluate to TRUE. In this example, we injected `' or 1=1 - - '`. This is a tautology. There is existing SQL in the app before the single quote that we can guess to be some sort of evaluation given that it's a login form. We are appending a tautology to the end of the statement by adding the `' or 1=1` so that it always evaluates to true. It's interesting to note here that it's the structure that matters and not so much the exact data. If you were to create an IPS signature for this attack, it will catch only variations using these exact values (`' or 1=1 - -`). But what about the following permutations?

```
' or 2=2 - -

' or J123=J123 - -

' or gonzo=gonzo - -
```

Attackers can easily evade signature-based protection by using techniques such as encoding, whitespace diversity, and IP fragmentation and TCP segmentation. We looked at IP fragmentation earlier in the text, so let's briefly explore how attackers can use encoding and whitespace diversity to evade signature-based detection.

Encoding and Whitespace Diversity

Web-based communications are commonly encoded and decoded in various different formats such as URL Encoding and UTF-8.

The reason for URL encoding to begin with is that URLs often contain characters outside of the ASCII character set. URLs are not allowed to contain spaces and this information is replaced with an intermediate identifier. For instance, a plus sign (+) would be represented by %20. Although there are legitimate reasons to use URL encoding, there are equally as many illegitimate reasons to do so. Attackers can use URL encoding to their advantage to obfuscate attacks so that their literal URL-encoded equivalents are not detected via attack signatures.

Simply put, whitespace manipulation involves adding additional whitespace between SQL statements to be injected in a form or some other type of web input. Many times, signature-only based solutions will be looking for a single space between SQL terms such as the SQL instructions UNION and SELECT.

Core WAF Capabilities

Signature-only approaches are prone to evasion and false negatives. This is why heuristics and rulesets are important. They analyze data structures as opposed to an exact replica of a dataset. A WAF ruleset in this case would be looking for the existence of a tautology no matter the specific values that make up the tautology.

WAF Rulesets and Heuristics to the Rescue

Keep in mind that the WAF is detecting input sent to the web server, not the communication from the web server back to the database. This is an important distinction. Database firewalls are pieces of technology that typically sit between the web server and the database. They are examining fundamentally different things. Database firewalls are analyzing SQLNet-type traffic and analyzing SQL

grammar structures for irregularities or malicious behavior. WAFs are also looking for irregularities and malicious behavior, but they are between the user (the internet at large) and the web server. Therefore, WAFs are particularly good at inspecting HTTP traffic that might indicate SQL injection based on HTTP specific messaging and artifacts such as POSTS, GETS, URL formats, and cookies.

Although WAFs can use blacklist rulesets and even in some cases actual threat signatures, you can also train them through learning or *baselining*. WAF baselining involves setting up a WAF in a safe environment and analyzing normal traffic over an extended period of time. This allows the WAF to minimize false positives through behavioral-based whitelisting.

WAFs will generally have rulesets that you can configure to look for and consider information such as the following:

- Country of origin
- Length of parts of the request
- SQL code that is likely to be malicious
- Strings that appear in requests

You can combine these conditions to form rules and rulesets, which are used to examine potentially malicious HTTP communication patterns.

XSS Attacks and WAF Protection

XSS attacks continue to dominate the OWASP Top 10. They represent critical risk exposure to businesses and consumers. As with most vulnerabilities, you can nip XSS in the bud during development, before it ever reaches production. The golden rule with injection attacks is to never implicitly trust user input. Developers should perform input validation and use output encoding. The reason to utilize output encoding is to modify untrusted input into a safe form where the data that is input into the system is rendered as data to the user without executing as code in the browser.

If your organization is responsible for developing the code in question, by all means, you should be incorporating these best practices into your software development life cycle (SDLC). But as you know, even when writing your own software, many times third-party libraries are included which can be inherently vulnerable. Also, if you are

using cloud software or shrink-wrapped software, you do not control the software development process.

With this awareness of development best practices, it's likely that you have a greater respect for the old adage "an ounce of prevention is worth a pound of cure." Now, let's turn our attention toward detection and correction.

Anatomy of an XSS Attack

The first step for an attacker is not to exploit an XSS vulnerability, but to find the vulnerability to begin with. Like all hacks, the first phase involves scanning and enumeration of targets. XSS vulnerabilities are a function of input that has not been sanitized properly before execution. Attackers can use tools such as web application vulnerability scanners and fuzzers to automatically find these types of vulnerabilities. Skilled hackers can also try various XSS attack techniques by hand. Generally, hackers and pen testers will use a tool for a first pass to find targets and then begin working "by hand."

Let's walk through a real-world XSS attack example, again using the Mutillidae vulnerable web server.

The page shown in Figure 3-5 is vulnerable to an XSS attack. The form accepts input to add to the blog but it does so without being properly encoded. In this case, the attacker can inject XSS or HTML into the input and break out of the context of the page. This type of attack is a *persistent XSS attack* because the injected JavaScript is being written and saved to the database as a blog entry.

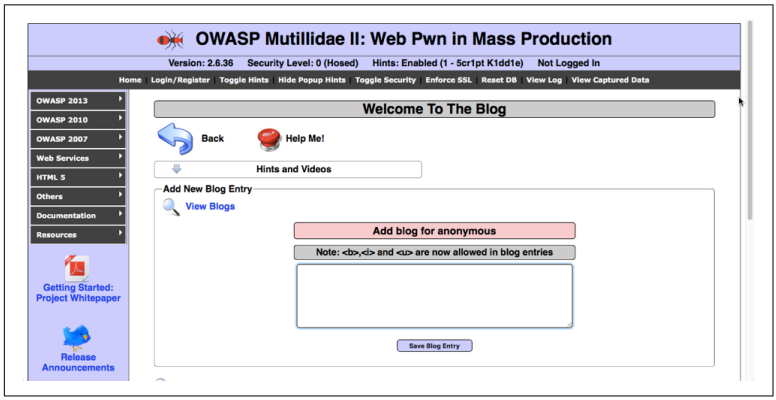


Figure 3-5. This form's input is vulnerable to XSS attacks

In Figure 3-6, we inject the following script:

```
<script>alert("XSS Attack")</script>
```

Nothing happens immediately when saving the blog entry, but look what happens when we try to make a new entry.

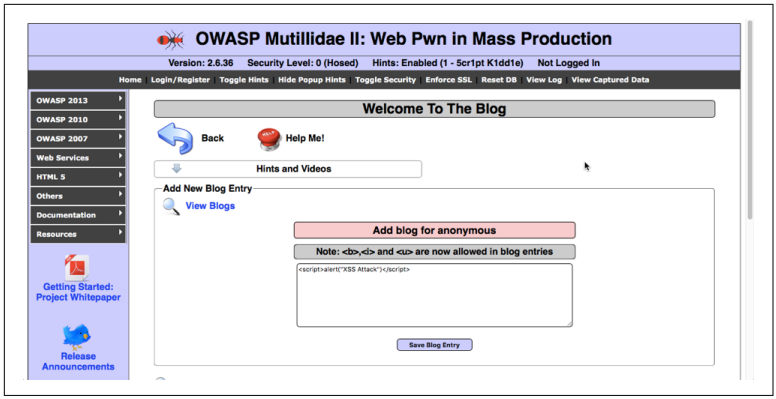


Figure 3-6. JavaScript is entered into the form's input

Figure 3-7 demonstrates that this is not a particularly vicious example, but keep in mind that almost anything that you can code as working JavaScript can be fair game after you've identified a vulnerable input.

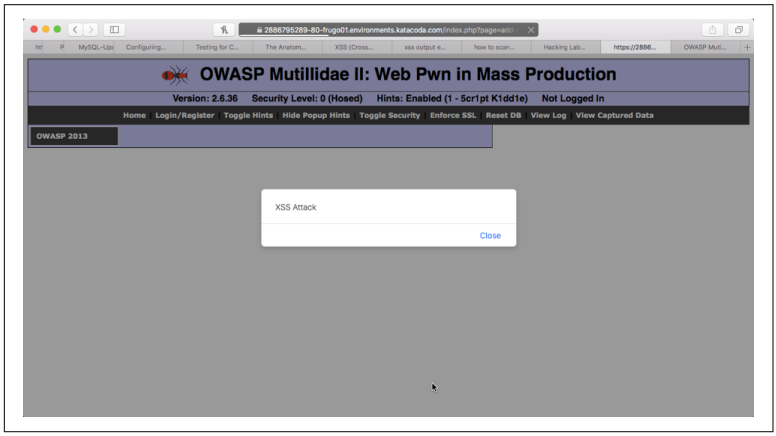


Figure 3-7. A successful persistent XSS attack. The JavaScript was injected and saved to the database as a blog entry.

WAF XSS Filters and Rules

As the examples in this section show, signature-based approaches are not particularly effective for XSS attacks. The signature would need to account for any JavaScript an attacker could dream up and place into the input. Instead, a different approach is required. It's about looking for patterns and behaviors as opposed to exact dataset matches. WAF rules come preconfigured to detect and block XSS patterns in the URI, query string, or body of a request.

How WAFs Can Protect Against Session Attacks

Session tampering is called out in the OWASP Top 10 and has been for a number of years. Any attack that can manipulate session data—such as cookie, parameter, and session ID tampering—can allow attackers to either inject themselves into an existing session (Man-in-the-Middle attack) or start a new session by replaying existing session information (session replay attack).

Not all WAFs are created equal, and some use different approaches in terms of the protections they provide. First, remember that a WAF sits between the requestor and the web server and functions as a *reverse-proxy*. A proxy by definition accepts a request on behalf of one entity and presents that request to the originally intended recipient on behalf of the original requestor.

Some WAFs can digitally sign artifacts such as cookies sent by web servers before delivering them to a client's browser. When a subsequent request from the client sends the cookie back to the server, the WAF can intercept the request and verify the signature. This is an interesting approach toward addressing cookie tampering.

Another consideration is to ensure that users are communicating only with servers that have valid digital certificates and that communications are HTTP over Transport Layer Security (TLS) and terminating at the WAF. This provides the dual benefit of protecting traffic in transit and decrypting it at the WAF so that the application layer traffic can be fully inspected.

Minimizing WAF Performance Impact

WAFs are devices that are deployed inline. This means that they are directly in the line of traffic. All traffic leaving and entering the net-

work must go through the WAF. This puts the WAF in the critical path, so to speak. For all the good that can be accomplished in terms of attack mitigation, a lot of drawbacks can be induced if the WAF is not engineered, designed, and deployed properly.

WAFs are processing much more data than a typical Layer 2/Layer 3 switch or router. They are processing data at every layer of the OSI model with a detailed emphasis on Layer 7 traffic. Think about all of the rules that we've talked about up to this point. They all need to be processed for every single packet. The opportunity to introduce incremental latency is immense.

So, how can you ensure that you not only choose the correct WAF technology, but design and deploy it correctly? Let's begin with the types of technology and capabilities that should be engineered into the WAF itself in order to optimize performance.

WAF Performance Optimization

Modern WAFs should be equipped to match or outpace the speeds of the Layer 2/Layer 3 devices that feed them. This means 10 GB throughput is expected. Customers expect WAFs to adapt to their network architecture.

Your WAF solution should support deployment modalities such as the following:

- Inline routing and Network Address Translation (NAT)
- Inline bridging
- Hanging off of a span port
- Reverse-proxy/SSL termination point
- Monitoring-only mode
- Blocking mode

Such solutions can accommodate multigigabit throughput and tens of thousands of transactions per second. Low latency SecureSphere appliances can manage heavy traffic loads without affecting application or network performance.

WAFs often function as SSL termination points. This is because traffic needs to be decrypted for inspection at the firewall before being allowed to ingress or egress the network. Encryption is a compute-

sensitive operation. Many modern WAFs support native *SSL acceleration* modules.

SSL acceleration is a means of offloading processor-intensive public-key operations for TLS to specialized hardware. These are often deployed as *SSL Accelerator Cards* that plug in to a bus slot of a device. Many SSL accelerator cards use Application-Specific Integrated Circuit (ASIC) or Reduced Instruction Set Computer (RISC) chips to perform some of the most difficult computational operations.

WAFs should also support expansion capabilities specific to HSMs for key storage and key management as well as network support for Fibre Channel interfaces.

WAF High-Availability Architecture

As you examine high availability (HA) architecture for a WAF solution you need to start with the individual appliance itself. It's important that the components within the appliance are fault tolerant from the outset.

Components within the appliance that should exhibit HA include the following:

- Power supplies (hot swappable)
- Hard drives
- Internal network connections

After you address HA within the context of the device itself, HA across devices is required. WAF deployments should support multiple horizontally scheduled devices to not only provide for HA, but to allow for sufficient horizontal scaling to accommodate any required network throughput.

WAF Management Plane

All WAFs require a *management plane*. This is typically a group or cluster of management servers that have exclusive and protected access to the WAF devices in order to manage them. Generally, the management plane is decoupled from the WAF devices themselves for performance and security reasons.

The WAF management plane should be secure and unify auditing, reporting, and logging for a given WAF deployment. You should be able to visualize security status and monitor incidents in real time to keep track of threats in your network.

In enterprise environments, many times multiple groupings of WAFs will be deployed across physical sites. Each of these sites should support their own "local" management server, but the solution should also account for holistic management via a "master" management server. By having a master management server, you can benefit from aggregated and consolidated views of your entire WAF deployment footprint and security posture. Think of it as being able to examine threat activities and trends across all sites through a single window. The WAF management plane should support granular Role-Based Access Control (RBAC) for administrators. Your chosen vendor should provide out-of-the-box support for predefined and custom reports for security and compliance. Additionally, you will need facilities to create custom reports using the data collected by your fleet of WAF appliances.

You can use master management servers to distribute security and audit policies to all WAF deployments across an enterprise via a centralized console. Having a centralized view allows security auditors and engineering teams to examine the security policies for all WAF deployments and perform policy configuration analysis to ensure continuity of security posture across the enterprise. Centralization of the management plane also allows for central monitoring of the health of the entire WAF fleet across multiple datacenters.

Emergent WAF Capabilities

As technologies advance, attackers continue to take advantage of new capabilities to advance their agendas. WAFs should be deployed as part of a defense-in-depth strategy and represent part of the overall security solution. As such, WAF vendors are starting to add integrations with adjacent solutions and incorporate WAF technology into existing technology trends such as DevOps, Security Information and Event Management Strategy, containerization, cloud, and artificial intelligence.

A common theme among emergent WAF capabilities is the march up the stack. By that I mean evolving from detecting and mitigating

purely technical attacks to address complex business logic–oriented attacks and fraud scenarios.

Security Information and Event Management Integration

Security Information and Event Management Systems (SIEMs) have been around for a number of years now. SIEMs gobble up information from every source they can attach to in order to consume, synthesize, correlate, and report on security events across a company's technology landscape. As part of Requests for Proposals (RFPs), customers have been consistently asking for their WAF solutions to cleanly and easily integrate with their existing SIEM solutions.

By integrating WAF security data into your company's existing SIEM, you do not need to log in to the WAF's interface to view the WAF logs. Additionally, by integrating these logs with your existing SIEM, you can correlate the WAF logs with other events across the enterprise to look for "kill chain" patterns and existing threats already lurking in your network.

DevOps Security Testing

It's true that the DevOps phenomenon (automated code deployments), if done correctly, can help to better secure and remediate in-house developed code on a continuous basis. DevOps environments incorporate continuous integration and continuous deployment (CI/CD) pipelines using tools such as Jenkins and Chef (among others) to continually release incremental software updates. Sometimes, this might be dozens of changes or more per day.

What's interesting about this model is that in standard DevOps environments, orchestrated containerization is utilized, which helps to compartmentalize and automate the instantiation and destruction of these updated microservices via *repaving*. In this paradigm, the container is immutable or read-only. There is no patching of the container. If a container needs to be patched, the DevOps pipeline just kicks off again, creating a patched version of the microservice, destroying the old microservice container and deploying the new one in the blink of an eye.

You might think that this solves our application security challenges completely. Even though it certainly represents a huge advancement

in application security, the challenge still remains that developers heavily use third-party code and libraries to develop their apps/microservices. Each third-party library represents a potential set of vulnerabilities. What if the developer is using an older version of the library that is vulnerable? Having a WAF infrastructure in place helps to address these gaps by providing security services during the timeframe between when the vulnerability is discovered and published and when the DevOps team repaves a new set of containers for the affected microservice.

During this window of exposure, the WAF can provide *virtual patching* and detection for known and unknown exploits through baseline behavior analysis and heuristics/ruleset analysis. Let me pause here and state that I'm making a *huge* assumption that the security team is even aware of the third-party libraries the development teams used to create their apps to begin with.

Although the window of vulnerability is smaller than the standard 30-day patch window of old, there is still a window and if the vulnerability is in a well-known third-party library such as OpenSSL (think *Heartbleed*), you can bet that attackers will be scanning your website for these vulnerabilities in short order to see if they can penetrate your networks and systems.

Security Operation Center Automation

Security Operation Centers (SOCs) are essentially call centers and operational hubs for monitoring and escalating security incidents. Although the concept of SoCs is not new, the way in which they are utilized and the issues that they face today have changed significantly over the past decade.

Some organizations outsource part of their SOCs to external entities that sift through large stacks of alerts and notify the company's IT department if the external entity believes there is a discernable issue that needs to be addressed. Other organizations run their SOCs completely in-house.

Here are a few key trends that have affected SOCs and their effectiveness over the past decade:

1. Consumerization of IT
2. Eroding perimeters

3. Software as a Service (SaaS)
4. Cloud computing
5. Industrialization of attacks
6. Big data (machine-generated information)
7. The Internet of Things (IoT)

We will address each one of these topics individually so that you can understand the respective impact that each of these trends has on SOCs.

Consumerization of IT and mobile

Mobile devices have permeated IT departments and BYOD (Bring Your Own Device) has become the norm. This means that IT departments no longer control devices that are on your network. Essentially all devices should be treated as "untrusted." You don't know if they are running antivirus, if they have been patched, or what software is running on them.

The challenge for SOCs in this case is that they no longer have agents loaded on many of these devices to monitor for attacks, vulnerabilities, or threats. It's almost impossible for an SOC to distinguish the difference between a trusted and nontrusted device, therefore they have all become untrusted out of necessity.

Eroding perimeters

Along with mobility comes transparent access to services. Wireless networking inside and outside of corporate network boundaries is a basic right and expectation for end users and customers. Customer traffic is not always traversing through well-defined corporate boundaries and DMZs.

For SOCs, this means that anyone can be accessing any service offered by the company from anywhere at any given time. In many cases, gone are the days of Virtual Private Network (VPN) access requirements to access company information. For SOCs to make sense of this, the level of automation and intelligence needs to increase exponentially.

SaaS

SaaS has really hit mainstream. As a result, companies are less likely to choose shrink-wrapped software packages and deploy them on-premises. The benefits of SaaS for business is that it is always updated and users have instant access to the latest features, anytime and anywhere.

Because SaaS is cloud based by nature, SOCs often don't have visibility into cloud services and cloud service activities.

Cloud computing

In this case, I'm referring to Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) types of services. Many IT departments are deploying new projects and migrating existing on-premises infrastructure to cloud infrastructure providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

For SOCs, this requires a shift in architecture and approach in order to have visibility into and control of IaaS- and PaaS-related services. The *cloud control plane* is also a new element that SoCs need to take into account from a monitoring and prevention standpoint.

Industrialization of attacks

When you combine the disparate location of users, devices, and services with the advent of industrialized botnets, you have a battlefield that spans the globe. For SOCs to keep track of and make sense of these activities in a meaningful way is an entirely new challenge.

Big data and IoT

In this instance, I'm referring to machine-generated data. The number of devices mapped to users has ballooned because of mobile and IoT devices. The average home can contain upward of 30 internet-connected devices, including gaming consoles, phones for each family member, voice assistant devices such as Amazon Echo dots in every room of your home, multiple laptops, smart home devices, and tablets. These are devices that are typically mapped to humans and human usage in some way. Now take into consideration all of the non-human mapped devices that exist, such as IP cameras, industrial sensors, switches, routers, and cars. It's estimated that by the year 2020, the internet will be populated by more than eight

billion devices. IoT traffic will overtake user-driven traffic by a significant margin in the span of the next several years.

For SOCs, the number of haystacks in which needles need to be identified is increasing exponentially. Outdated approaches to identifying threats need to be modernized.

Cybersecurity Skills Shortage

It's estimated that, as of 2017, there are more than a million unfilled cybersecurity positions globally. SOCs are understaffed and short on skills.

WAFs and Their Part in SOC Modernization

For chief information security officers (CISOs), automation is the name of the game. With the sheer increase in the number of devices and the diversity of access, having humans scour tens of thousands of alerts is no longer effective or even meaningful. High false-positive alert rates only compound the issue for alert-weary SOCs.

SOC activities require increased automation to have any hope of keeping up. The first step is to apply concepts such as machine learning to capture baselines and identify deviations from normal traffic. This has become a base requirement. There simply aren't enough skilled SOC analysts to eyeball every machine-generated log event that scrolls across their monitors, nor would it be desirable to do so even if you could.

Event identification requires machine learning and adaptation along with the application of contextual information such as identity data.

Modern WAFs can assist in addressing these challenges by taking advantage of technologies such as machine learning, threat intelligence, and threat-feed correlation capabilities. By integrating with external systems, WAFs can help to drive action via *security incident response orchestration* platforms.

WAF Threat Intelligence and Feed Correlation

To help beleaguered SOC Managers, WAF solution providers have incorporated the use of crowd-sourced threat intelligence. Given the exponential set of diverse data and threats, it helps when you can

coordinate with others to gather threat intelligence on a large scale and apply it locally as needed.

Crowd-sourcing threat intelligence borrows from principles associated with crowd-sourcing, social media, and the open source community. Fundamentally it is the sharing of security-related information such as vulnerabilities, reported security incidents, and code and solutions for addressing threats operationally.

Crowd-sourcing of threat intelligence takes place at government, organization, and vendor levels.

Some vendors collect and aggregate security-related events from their customers and partners.

As a result, organizations can subscribe to threat intelligence services such as these:

1. Reputation services
2. Bot protection
3. Account takeover protection
4. Fraud prevention
5. Emergency feeds—zero day

A given SOC could never hope to gather this data on its own, much less disseminate it and incorporate it operationally. Threat intelligence services allow them to incorporate this data to help facilitate more accurate alerting and drive meaningful automation within SOC environments.

General reputation services

At their core, reputation services are about tracking known bad IP address actors. Even though attackers can still anonymize via Tor (originally known as The Onion Router) and other proxies, often they do not. There are still an immense number of IP addresses that are used for direct attacks.

Although there are still a larger number of direct IP addresses used, the challenge is that many times they are dynamic in nature. They are reissued by DHCP service providers on a continuous basis. Often, public IP address lists are static in nature and are typically outdated.

As you look to subscribe to IP reputation services, make sure you are choosing a service that is dynamically updating its database continually. Typically, an IP reputation service will break down IP addresses into known threat categories such as Tor Proxies, Web Attackers, Spam Sources, Anonymous Proxies, Botnets, Phishers, and Scanners.

Bot protection feeds

Some threat intelligence feeds focus on botnets exclusively. These feeds focus on botnet command and control (C&C) networks. For instance, Microsoft provides a real-time threat intelligence feed specific to botnets as part of its botnet takedown operations. Its botnet feed includes three subfeeds. Customers can look for malware infections that accompany botnet activity or correlate host data with information on various internet scams or fraud activities.

Account takeover protection feeds

Account takeover attacks involve the use of compromised credential information. These credentials might have been compromised as part of a separate data breach and are sold or distributed on the dark web in exchange for Bitcoin or Ethereum payments. Currently it's estimated that a compromised account is worth roughly $3 per account.

For an attacker there is value in these accounts due to password and credential reuse. It is common practice for users to reuse the same credentials across multiple sites. Attackers in turn will feed this information into botnets to farm out account takeover activities at industrial scale, as illustrated in Figure 3-8.

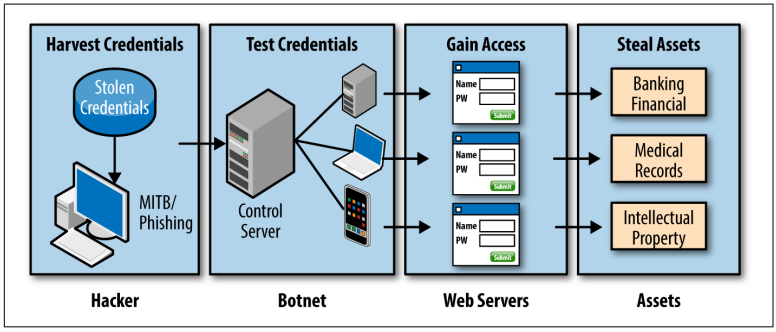


Figure 3-8. Anatomy of an account takeover attack

The first step for an attacker is to harvest credentials. This can take place via separate botnet-driven phishing campaigns. Attackers also can just pay for credentials using cryptocurrency via the dark web to outsource the process of credential gathering.

Next, the attacker will either use their own botnet or "rent-a-botnet" to carry out credential-stuffing attacks using the credentials they have collected either themselves or that they paid for on the dark web.

The attacker then uses their botnet (rented or otherwise) to carry out the credential-stuffing attacks against webservers. Upon successful attack, the hacker plunders the data and uses it for whatever purposes they see fit.

Some threat feeds focus exclusively on tracking account takeover activities such as the scenario illustrated in Figure 3-8. Your organization can then feed your security devices (such as WAFs) this information so that real-time threat filtering can take place using up-to-date threat information

Fraud prevention

Some vendors offer threat feeds that specifically track fraudulent activities. RSA, for example, has an intelligence operation comprising a dedicated team of analysts who monitor underground forums, IRC chat rooms, Open Source Intelligence (OSINT), and Dark Web channels and incorporate this data into its Fraud Intelligence Feeds subscription service.

The types of data RSA collects and disseminates include human intelligence, deep-web sources, and even ad hoc on-demand research services. All of this data represents information your security infrastructure and SOC team can consume, synthesize, and process to enhance your real-time operational security posture.

Emergency feeds—zero day

As the name suggests, these feeds focus on the distribution of newly discovered zero-day vulnerabilities so that your organization can prioritize efforts and implement compensating controls where possible. Zero-day threats are threats relative to vulnerable software that does not have patches available yet.

WAFs Authentication Capabilities

WAFs solutions let you implement strong two-factor authentication on any website or application without integration, coding, or software changes. Single-click activation lets you instantly protect administrative access, secure remote access to corporate web applications, and restrict access to a particular web page. Two-factor authentication manages and controls multiple logins across several websites in a centralized manner. Two-factor authentication is supported using either email, SMS, or Google Authenticator.

Malware Inspection and Sandboxing

Malware inspection and sandboxing is typically not handled by WAFs directly, for security and performance reasons. The purpose of malware sandboxing tools is to analyze files for unknown threats such as zero-day attacks or malware variants that do not have existing signatures.

It might seem like malware in the wild without signatures is uncommon, but it's not for several reasons. One reason it's alarmingly common is because attackers have tools such as packers and crypto tools with which they can repackage existing malware quickly in such a way that their signature becomes completely different. Attackers can do this repeatedly in an automated manner, even through the use of botnets. These scenarios can render antivirus and other signature-based/file scanning controls practically useless.

A primary use case for malware inspection and sandboxing technology is to supplement filtering for email gateways. Email security gateways are typically looking for malware that matches known signatures. Email security gateways can hand-off suspect traffic or even all attachments to a sandbox appliance for further inspection.

These appliances work by doing static and dynamic analysis of executables and monitoring how they interact with the VM in which they have been instantiated. They are looking for backdoor communications to botnet C&C, Windows registry manipulations, as well as any other observable behavior and create a profile and a signature for that particular executable. The sandboxing tool might flag it as malicious by way of heuristic analysis and/or it might hand it off to a human operator for further inspection.

One of the main reasons for having sandboxing tools is to allow incident response teams and forensic technicians a safe environment in which they can analyze the behavior of malware.

Malware sandboxes couple well with email gateways due to the asynchronous nature of email. Malware sandboxing and inspection takes time. It takes time to allow the malware to run and identify its behaviors.

From a WAF standpoint there is potential for integration, but the challenge here is that traffic flowing through a WAF is expected to preserve its real-time characteristics, so coupling WAFs with malware sandboxing technology is not always a practical alternative.

Detecting and Addressing WAF/IDS Evasion Techniques

As part of a WAF evaluation, you should test for core attack vector coverage. This includes XSS, SQL injection, file inclusion, and Remote Code Execution (RCE) testing.

When evaluating WAF technologies, you also want to ensure that you are evaluating how well the solution addresses WAF evasion techniques. Here are some examples of WAF evasion techniques:

- Multiparameter vectors
- Microsoft Unicode encoding
- Invalid characters
- SQL comments
- Redundant whitespace
- HTML encoding for XSS
- JavaScript escaping for XSS
- Hex encoding for XSS
- Character encoding for Directory Traversal

Make sure you know how the WAF handles known vulnerabilities, its false-positive, and false-negative rates, and incorporate these checks into your testing process.

Following are a few WAF bypass attempt examples:

Mixed case
: *http://website.com/index.php?page-id=28 uNiON sELeCT 4,8,7*

URL encoding
: *http%3A%2F%2Fwebsite.com%2Findex.php%3Fpage-id %3D28%26nbsp%3BuNiON%20sELeCT%204%2C8%2C7*

WAFs can serve as the best front-line defense for applications. Although WAFs are a robust addition to any defense-in-depth strategy, they are not foolproof in and of themselves. They are most effective when deployed along with complementary technologies.

Virtual Patching

Virtual patching capabilities are not entirely new in the WAF space, but there are new and emerging capabilities in this area.

Before we get ahead of ourselves, allow me to define what virtual patching actually is. Virtual patching is the quick development and short-term implementation of a security policy intended to prevent an exploit from being successfully executed against a vulnerable target.

As an example, a Common Vulnerability and Exposure (CVE) update outlines a specific WordPress form that is vulnerable to SQL injection. Basic WAF coverage would catch this by virtue of general coverage for SQL injection in all forms, against all parameters. A virtual patch, however, would be specific to the page, form, or parameter in question, and even the type of injection used.

Virtual patches can help to protect applications without modifying an application's actual source code. Virtual patches need to be installed only on the WAFs, not on every vulnerable device.

Vendors working in partnership with High-Tech Bridge can utilize proprietary machine learning technology to intelligently automate web vulnerability scanning. You can utilize this data to create dynamic and highly reliable virtual patching rulesets within the WAF to mitigate discovered vulnerabilities without even having to apply the actual patch.

Many times, a company's patching cycle falls within scheduled maintenance windows, which might be every weekend or just once a

month. The implementation of these virtual patching rulesets in the WAF provide interim compensating controls until the actual patches can be applied to the vulnerable systems.

Adjacent Solutions and Technologies

WAFs serve as an integral component for application security and should serve as one component of an overall application defense-in-depth strategy. In this section, I focus our attention on auxiliary technologies that complement WAF deployments and can help to solidify your organizations overall security posture.

API Gateways

The purpose of API gateways is to insulate and abstract internally APIs and allow them to be securely published to external consumers as part of the overall API economy. API gateways started out as Simple Object Access Protocol (SOAP)-oriented devices that would filter requests based on SOAP-oriented XML taxonomies and proxy requests to an internal web service protected on a private network segment. API gateway security policies could be configured to trigger security events based on triggers. These triggers could be a given URI structure, value, or traffic source. An example of a trigger might be to initiate an authentication request or to redact a dataset dynamically.

Often, the format of the requested URL is abstracted from the actual URL. For instance, a request for *https://website.com/service1* externally might map to *https://website.internal/registryservice* internally. Another benefit in abstraction is that a seemingly unorganized cadre of internal web services can be published and shared using an orderly URI structure for external consumption.

API gateways of course now support Representational state transfer (REST)-based requests, which have become the norm and perform functions similar to those described earlier but in a REST context.

API gateways can also do protocol translation. You can configure them to receive a REST request from the internet and translate that into a SOAP request for internal services that have not been fully migrated to REST. An API gateway could virtually present multiple REST and SOAP services as one logical REST service API. So, you can see there are lots of possibilities here and that API gateways per-

form very specific functions. Most of the functions involve proxying and translating between data formats. You can also use API gateways to inject security rules into communications. For instance, you might want to trigger an OAuth 2.0 client credential flow if a REST API is called with the format *https://website.com/newrestapiv2*, and if a client calls *https://website.com/newrestapiv1*, you might require only basic authentication over TLS.

API gateways are becoming an important component in serving content from microservices-based deployments. With multiple potentially disparate microservices, they can be served as on logical API.

Notice that this functionality is fundamentally different between what an API gateway does and what a WAF does. WAFs are inspecting Layer 7 traffic for malicious behavior, whereas API gateways are focused on serving microservices and doing Quality of Service (QoS) and enforcing security rules.

The point here is that they are complementary solutions. Architecturally, it's a best practice to have your API gateways tucked behind your WAFs so that Layer 7 API calls are properly terminated and inspected before reaching your API gateway. By the time the call reaches the actual internal microservice, it has been fully inspected for malicious intent and vetted by way of security rules such as authentication and authorization constructs.

Bot Management and Mitigation

Bots are not one size fits all. Any function you can imagine can be automated via bots and botnets. First of all, the term "bot" has really gotten a bad rap lately. All bots are not evil. A bot is really an automation of one or more tasks. On the internet, search engine *web crawlers* or bots have been around for more than 20 years. Search engine bots index billions of web pages continually to provide up-to-date snapshots of content on the web. Another type of "good" bot are chatbots. Chatbots can use discussion trees, AI, and Natural-Language Processing (NLP) to act as virtual assistants and facilitate Help desk automation.

It's estimated that almost 50% of all traffic on the web is bot driven. OWASP has categorized 21 types of bad bots (automated threats) described in a new taxonomy. Some of the most malicious bots are:

- OAT-001 Carding
- OAT-002 Token Cracking
- OAT-003 Ad Fraud
- OAT-004 Fingerprinting
- OAT-005 Scalping
- OAT-006 Expediting
- OAT-007 Credential Cracking
- OAT-008 Credential Stuffing
- OAT-009 CAPTCHA Defeat
- OAT-010 Card Cracking
- OAT-011 Scraping
- OAT-012 Cashing Out
- OAT-013 Sniping
- OAT-014 Vulnerability Scanning
- OAT-015 Denial of Service
- OAT-016 Skewing
- OAT-017 Spamming
- OAT-018 Footprinting
- OAT-019 Account Creation
- OAT-020 Account Aggregation
- OAT-021 Denial of Inventory

There are many more bots per the list but hopefully these examples give you a sense of the enormity and scope of the challenges when it comes to bot defense:

Scraper bots

Although search-engine bots are indexing the contents of the web for the purpose of search, there are many companies that are using bots to "scrape" content from sites and incorporate into their own sites. A prime example of this activity is price comparison sites which aggregate goods and services along with their respective price from multiple sources.

Captcha defeating bots
: Captcha defeating bots are adept at using machine learning and AI to solve and defeat Captchas. Of course, the entire point of Captchas is to keep the bots at bay. This is a true ongoing game of cat and mouse.

Credential-stuffing bots
: Credential-stuffing bots generate mass login attempts used to verify the validity of stolen username and password pairs.

Denial of Service (DoS) or Distributed Denial of Service (DDoS) bots
: These are the types of bots that generate DoS traffic from multiple disparate sources and geographies leveraging Zombies (compromised systems). Defending against DDoS attacks is traditionally very challenging because the IP addresses and sources are continually changing.

Sniping bots
: Do you ever wonder how someone ended up beating you on that last-minute bid for an item on eBay? It could be a sniping bot.

Carding bots
: These are bots that make multiple payment authorization attempts to verify the validity of bulk stolen payment card data. Those $1.01 charges you see could be a carding bot checking the validity of your credit card data.

WAFs can assist with certain types of bots such as credential-stuffing bots and scraping bots, but they might not be geared toward addressing the full breadth of aforementioned use cases and scenarios. A new class of bot mitigation and defense devices has been introduced into the market to specifically address newer and emerging bot threats, which might be useful in augmenting WAF bot mitigation capabilities for fringe use cases.

A few things to consider in this space is that it's not just websites that are being affected by this broader classification of bots: it's APIs, as well. Eventually addressing all vectors such as web, mobile, and API can help provide the broadest range of bot mitigation. Most bots are designed by their creators to evade detection, which means that detecting and mitigating them will now always be a futile endeavor.

Runtime Application Self-Protection

Runtime Application Self-Protection (RASP) is a new category of application defense technology that you can actually embed into an application's runtime. It essentially becomes part of an application's runtime by way of the underlying runtime framework. Current frameworks generally supported for this class of solution include CLR (.NET Framework) and the Java Virtual Machine (JVM). There are variations of RASP technology that allow for the embedding of RASP agents inside various application subcomponents such as controllers, business functions, data layer libraries, and any other application components. This essentially provides enhanced instrumentation (application logging capabilities), which you can use to provide deeper insight into specific activities within an application.

Some interesting applications of this technology could be for IoT devices that don't have the benefit of being behind a WAF. IoT devices are typically spread out over numerous untrusted networks. Having these types of protections embedded into these devices could help serve as a compensating control for IoT deployments.

RASPs can respond to runtime attacks via custom actions, which can include the following:

- Replacing tampered code with original code at runtime
- Safely exiting or terminating an app after a runtime attack has been identified
- Sending alerts to monitoring systems notifying of an application runtime attack

RASPs might or might not serve as a value-added complement to your existing WAF implementation. It will be dependent upon specific variables relative to your applications and network infrastructure.

Content Delivery Networks and DDoS Attacks

Content Delivery Networks (CDNs) consist of a geographically distributed network of proxy servers placed in datacenters throughout the world to distribute cached content and access controls closer to the users that consume them. The initial use cases were to cache

content and improve performance, but the role of CDNs has expanded to address advanced security capabilities for applications.

Initially CDNs were caching content for on-premises web server deployments exclusively. However, with the advent of cloud computing (SaaS, PaaS, IaaS, and otherwise), CDNs offer a unique topology to facilitate security enforcement for cloud environments.

Just to be clear, traditional WAF technology can be deployed in the form of a virtual appliance in the Amazon Elastic Compute Cloud (Amazon EC2) cloud, for example. And this might be suitable and appropriate for organizations that want this level of fine-grained control specific to deployment and configuration. CDNs are beginning to offer basic WAF functionality to address OWASP Top 10 vulnerabilities.

CDNs are especially well suited to addressing DDoS attacks. By providing cached content at the perimeter of the web, they can help to absorb the attacks and minimize the performance impact on the actual web servers responsible for serving the site itself.

To use CDN protection, you need to change your DNS records to ensure that all HTTP/S traffic to your domain is routed through the CDN network. As a result, the CDN masks your origin IP address and continually filters inbound traffic, blocking DDoS traffic while legitimate requests flow inbound unimpeded.

These CDN's are globally distributed in nature and are always-on solutions. You should compare the specifications of various providers and ask about Service-Level Agreements (SLAs) and inquire about the capacity of their networks to handle attack load.

CDNs can generally provide reports that you can consume that show the amount of traffic served by the CDN versus traffic that is served from your website.

Data Loss Prevention

Data Loss Prevention (DLP) solutions are tasked with ensuring that sensitive data doesn't leak out of corporate boundaries. Legacy DLP solutions functioned by filtering content at traditional network perimeter enforcement points and email gateways. With the advent of BYOD and cloud computing, perimeter-only enforcement models are not as effective as they once where for preventing data leakage.

Modern DLP solutions expand beyond the perimeter and integrate with cloud providers and directly with user devices. This means that there is often an agent running on devices that inspects traffic looking for sensitive data leakage. From a cloud perspective, DLP vendors are beginning to create plug-ins that inspect cloud activities and function as monitoring and enforcement points alongside various cloud components.

Let's look at a concrete example of how modern DLP might be deployed today at a given company. Company X used Microsoft Office 365 for email, and Dropbox for its file storage. Its users bring their own devices and access corporate on-premises resources and cloud-based resources such as Office 365 and Dropbox from any network.

In this instance, DLP agents have been deployed to mobile devices and laptops that request access to corporate resources. They are not allowed to access resources without having the DLP agent installed. The chosen DLP has capabilities to monitor cloud solutions such as Dropbox and Office 365 email to inspect for data leakage from the cloud, as well.

There are many different vendor options and trade-offs when choosing DLP solutions. Some considerations include the need for agent software on mobile and other BYOD devices such as tablets and laptops. The ability to address cloud services utilized by your company is another important consideration.

Other capabilities to look for in a DLP solution include the ability to effectively scan unstructured data across numerous devices and cloud environments and identify sensitive data types. Deep email provider integration is a key requirement for this class of solution.

DLPs are complementary to WAFs. Whereas WAFs, which are either deployed in the cloud or on-premises, are looking for web application threats, DLPs are looking for the egress of sensitive data outside of corporate boundaries. These corporate boundaries can be as fine grained as an employee's BYOD device, the corporate network, or various cloud services consumed by the company and its employees.

Data Masking and Redaction

Data masking and redaction solutions are intended to conceal data or redact it so that only those who have a need to know can see the full dataset. Everyday examples of redaction include the redaction of social security numbers or credit card data. Following is an example:

```
SSN – 530-**-****

CC - *****-*****-*****-4238
```

In many cases, the full dataset is not redacted so that the data still has some meaning in terms of verification. For instance, when you talk to a customer service representative on the phone and they ask you for the last four digits of your social security number, they are likely looking at a computer screen that has redacted all of your social security number except for those last four digits. The benefit here is that it can be used as part of a series of identifying questions to ensure that you are indeed the person who owns the account in question.

There are several key modalities in which redaction technologies are most often employed:

- Real-time application-based redaction
- Production to development data masking and redaction
- Document-based redaction and masking

The preceding example in which the customer service representative sees a redacted set of data is a good illustration of the first modality, real-time application-based redaction.

The end goal of the second use case is to allow developers to be able to import production datasets such that the sensitive data is masked and that the remaining datasets retain their context for application testing purposes. These solutions will typically be incorporated as part of a data processing pipeline, copying data from production databases to development environments.

The last example is document-based redaction and masking. Here, we are referring to unstructured data such as word documents and spreadsheets. There are a variety of solutions in this space that either enforce the redaction at the network level, at the client level, or via a combination of both.

Some WAFs have the ability to provide some level of data redaction, but it is not considered to be a core competency of the WAF space. Redaction solutions typically complement WAFs for finer-grained DLP use cases and requirements.

WAF Deployment Models

WAFs serve as an integral component for application security and should serve as one component of an overall application defense-in-depth strategy. In this section, I focus our attention on auxiliary technologies that complement WAF deployments and will help to solidify your organization's overall security posture.

On-Premises

On-premises deployments are the traditional type of deployment for WAFs. Within the context of an on-premises deployment, there are several operational or networking modes in which most WAFs can be configured to operate.

Native Cloud

Native cloud–based WAFs are sometimes an extension of existing CDNs or offered as primary distributed/cloud-based security offerings. There are many benefits to using WAF-as-a-Service.

WAF-as-a-Service doesn't require you to deploy any hardware or software; it is simply consumed as a cloud-based service. Setup typically involves manipulation of your DNS records so that they point to the WAF cloud services. The WAF cloud services will in-turn proxy back to your actual web properties.

WAF-as-a-Service offers performance benefits in that you can use WAFs that are closer to the requestor and minimize network round-trip latency.

Cloud-Virtual

Cloud deployment models for WAFs share more similarities than differences. Particularly in IaaS deployments. In an IaaS deployment model, instead of a physical appliance, the WAF is deployed as a software appliance or VM.

In this deployment model, most deployment modes are available according to the configurability of the underlying IaaS virtual networking platform. The standard, proxy/router-based WAF deployment is generally supported in most IaaS clouds such as AWS and Azure among others.

In-Line Reverse-Proxy

An in-line reverse-proxy is perhaps what most people think of when they think about a firewall deployment. This type of deployment uses NAT for address translation and proxies traffic between internal and external networks. By virtue of NAT, IP addresses in the internal network are hidden from the outside world. As a proxy, the WAF directly intercepts all traffic and is fully in-line.

In this mode, no traffic can bypass the WAF. All traffic for the configured network segments ingress and egress through the WAF.

Within the context of this deployment mode there are several different deployment alternatives. Some customers choose to use a three-legged model in which the WAF has three interfaces including a public interface, a screened subnet interface, and an internal interface.

The benefits of this type of model are that it takes less hardware, but there are drawbacks. The first drawback in this type of deployment would be availability. The three-legged model is usually a model with only one WAF. No HA is included in this type of model.

Another drawback of the three-legged model is that because a single device is partitioning the public network, the screened subnet, and the internal network, one wrong logical configuration in that single WAF can expose the entire screened subnet or even the internal network.

A tried-and-true WAF deployment model is what is sometimes called a *firewall sandwich*. This is a purist DMZ architecture. In this design, there is an external-facing WAF and a separate internal-facing WAF. The network segment(s) between the two WAFs becomes a true DMZ. A best practice in this model is to deploy the outer WAFs as HA, and the inner WAFs as HA.

Benefits of this model include the fact that a change on one firewall doesn't necessarily compromise the internal network. Sometimes in this model, the outer firewalls will be used as an SSL/TLS termina-

tion point, whereas others choose to place load balancers in the DMZ and terminate SSL there. By terminating SSL at the WAF, the WAF has the opportunity to inspect the HTTP traffic. If the SSL/TLS traffic is terminated at a load balancer within the DMZ, at least the inner WAF has a chance to inspect it. Some network architects decide to place load balancers in front of the outer WAFs and terminate TLS/SSL there before traversing the outer WAF pair.

Transparent Proxy/Network Bridge

In bridging mode, a WAF is deployed as a transparent Layer 2 switch on the network. This deployment mode offers high performance and requires no changes to web applications or the network.

You can also deploy some WAFs in a routing mode. This mode is best if NAT is needed or if IP addresses are in a different subnet than other portions of the network.

Out of Band

WAFs can also be deployed out of band. This means that they are not deployed directly in the traffic stream. In this type of deployment, a WAF could be connected to mirrored span ports or taps that duplicate traffic off of the wire and direct it to the WAF for passive processing.

A benefit of this mode is that false positives do not drop network traffic. One of the drawbacks is that attacks will pass through the network without being blocked.

Multitenancy

Multitenancy support is an important issue given the shared nature of cloud computing.

Companies that consume IaaS services need WAF solutions that can help protect them from other cloud tenants and from external attacks.

Cloud service providers are eager to address multitenancy in order to drive down cost. Cloud service providers are interested in sharing network, compute, and database resources where possible between customers. The potential for security issues increases exponentially in multitenant environments. Cloud-deployed WAF software appli-

ances can be used to help mitigate application-based attacks that might try to steal data from multitenant systems.

Architecturally, you can choose to deploy multiple WAFs or use a shared WAF architecture to protect multitenant environments.

Single Tenancy

Single-tenant environments can be addressed directly with a standard WAF deployment. WAFs can be deployed as a firewall sandwich or with a screened-subnet model, as referenced previously. If the applications are mission critical, I recommend that you deploy two or more WAFs in a clustered/load balanced pair for each tier to ensure HA.

Software Appliance Based

Many WAFs are typically deployed as hardware-based appliances but also support deployment as a VM or software-based appliance. In this model, the WAF software is preloaded and configured into a VM image that is deployed and configured much like the hardware version of the appliance.

Software appliance–based WAFs offer quick setup time and flexibility when it comes to deployment options.

Hybrid

Hybrid solutions are becoming the norm. Almost all organizations have a mix of cloud and on-premises IT. Companies need solutions that can address various scenarios. In Chapter 4, I cover how you can use combinations of these design architectures, solutions, and deployment models to address specific business requirements.

CHAPTER 4

Designing a Comprehensive Network Security Solution

In this last chapter, we cover a use case and take what you've learned throughout this book to apply a comprehensive solution to a particular set of business requirements for a fictitious company, XYZ Corp.

XYZ Corp

XYZ Corp is a holding company that has traditionally had on-premises computing assets in the form of web servers. XYZ is a reseller of online advertising. As such, it hasn't really traditionally stored much sensitive information directly about customers. XYZ Corp is diversifying within the marketing space and has recently purchased a direct online advertiser.

From a business standpoint this means that the type of data that will be processed by XYZ Corp will include customer information across multiple geographies. The company needs to scale its advertising campaigns quickly so that it is expanding into the cloud to take advantage of IaaS services.

XYZ Corp has asked you come up with a security architecture that addresses its existing on-premises assets as well as its new cloud footprint. The company also needs the architecture to take into account the protection of sensitive information, which it is processing as part of its direct marketing campaigns. XYZ Corp advertising

campaigns require HA because they are time sensitive. It plans to run its own email infrastructure to facilitate these campaigns. The company will also be using some third-party APIs for SMS and other messaging platforms.

From an architectural perspective, let's distill XYZ Corp's requirements into some key considerations:

1. Data security
2. Hybrid cloud deployment
3. Heavy mail usage
4. API integration
5. Cloud/internet presence—protection against DDoS and bots
6. Global audience
7. Data is subject to regulation such as the EU's General Data Protection Regulation (GDPR)

Let's distill the company's requirements into a core WAF architecture, as referenced in Figure 4-1.

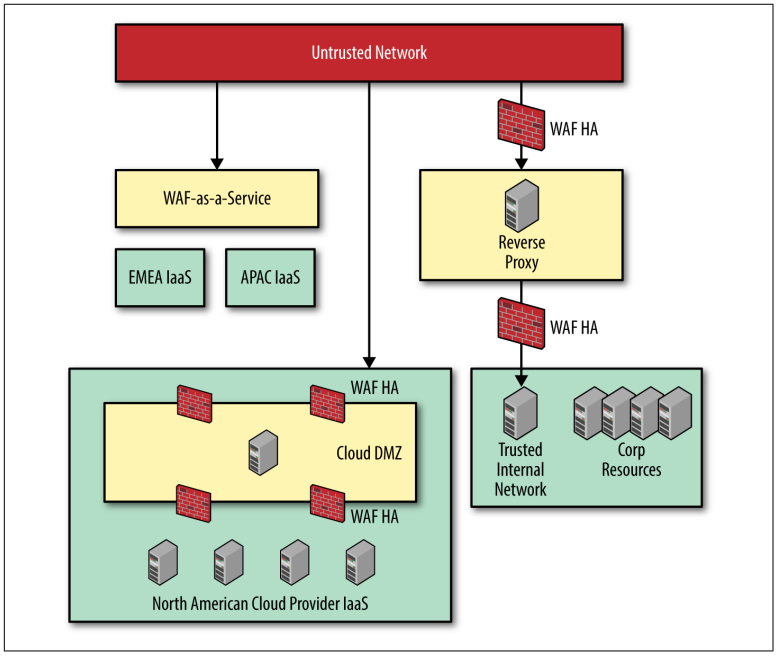


Figure 4-1. Core WAF architecture

Figure 4-1 illustrates that we have an HA WAF sandwich–style architecture for both the on-premises deployment and the cloud environment. XYZ Corp wants HA, and it needs the highest levels of security for both its cloud and on-premises infrastructure.

Although the architecture looks very similar between both the on-premises and cloud environments, there are some semantic differences. The on-premises architecture is using physical WAF appliances, whereas the cloud deployment is utilizing software-based WAF appliances. Additionally, XYZ Corp has determined that it will need smaller cloud points-of-presence in the APAC and EMEA regions. These are not large enough to necessarily justify the overhead of procuring and managing WAFs, so XYZ Corp decided to use WAF-as-a-Service for these IaaS clouds.

So, how do we begin to achieve and address the original business requirements with this core component of the security architecture?

1. Data security: SQL injection, web app protection—OWASP Top 10
2. Hybrid cloud deployment: software-based appliance for cloud, hardware-based appliance for on-premises, and WAF-as-a-Service for smaller cloud points-of-presence in EMEA and APAC
3. Heavy mail usage: TBD
4. API integration: TBD
5. Cloud/internet presence: software-based appliance for cloud and hardware-based appliance for on-premises—protection against DDoS and bot
6. Global audience: software-based appliance for cloud, hardware-based appliance for on-premises, and WAF-as-a-Service for smaller cloud points-of-presence in EMEA and APAC
7. Data subject to regulation such as the EU's General Data Protection Regulation (GDPR): OWASP application protection

Although WAFs address many of these business security requirements, there are some that they do not address directly. This will require you to build off of your WAF core and augment with some adjacent services.

You still have some outlying requirements to address. Specifically, the API and email-specific requirements. Why don't the WAFs address these directly? Well, as I mentioned earlier, WAFs are very HTTP specific. With email, we are talking about SMTP and a mix of other potential protocols such as IMAP and POP3. And although the APIs are largely going to be SOAP or REST oriented over HTTP, the handling of API traffic is better suited to API gateway–specific devices. API gateway devices recognize REST and SOAP formats and act as an enforcement point and abstraction layer for web-based API calls.

So, XYZ Corp could go in a few different directions to address these adjacent issues. It could use some sort of on-premises email gateway filtering solution or outsource it altogether as a service. XYZ Corp decides to outsource its email filtering and protection to a third-party provider.

For API gateway services, the company decides to deploy API gateways in an HA configuration across its various points of presence. It places these HA API gateway pairs in the DMZs of its points-of-presence, respectively.

For API gateway services, XYZ Corp decides to deploy API gateways in an HA configuration across its various points of presence. It places these HA API gateway pairs in the DMZs of their points-of-presence, respectively, as illustrated in Figure 4-2.

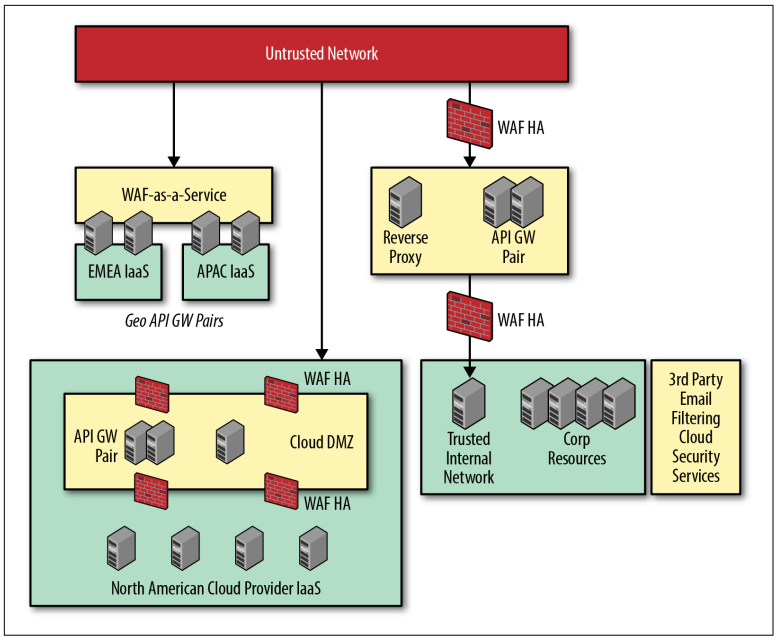


Figure 4-2. Core WAF and adjacent technologies

A Note About Native Cloud Security Services versus Specialized Services

Big cloud providers such as AWS have begun to offer additional security services as part of its traditional IaaS offerings. For instance, AWS offers an API gateway service and a WAF service.

Simply put, these are basic services. I encourage you to perform you own analysis, but generally speaking, they address some of the lowest common denominators in terms of threat protection. Sometimes, these solutions are viewed as a "good-enough" or "better-than-nothing" approach. They lack many of the advanced features such as protection against DDoS, bot, and account takeover, among others.

Afterword

The security threat landscape is changing as fast as the technology that is used to innovate computing itself. It's critical that companies take advantage of emerging technologies such as artificial intelligence, machine learning, and cloud services to not only advance and innovate, but to keep up with the bad guys. Bad actors are adopting artificial intelligence and machine learning and incorporating it into their botnet frameworks. This means more automated attacks at greater volume and with greater precision and sophistication. Deploying the appropriate WAF components in the proper way across hybrid cloud deployments along with other adjacent technologies can help ensure that you are designing and deploying modernized security solutions that keep up with the advancements across the technology landscape.

About the Author

Chad Russell has over 15 years of security experience, managed teams of security engineers and analysts for an internet banking provider, and acted as a security consultant working for companies including SAP, Microsoft, and Oracle. Currently, Chad leads and conducts security risk assessments for customers throughout North America with an emphasis on cloud security, identity governance, network security, social engineering, mobile security, breach assessments, database security, and access management.

Chad also serves as the chief cybersecurity training director for Web of Security. Web of Security consults with IT security managers and chief information security officers to help them assess and address areas of cybersecurity need. The training programs focus on SOC, Blue Team, Red Team and CISO enablement. Training modalities offered by Web Of Security include live/online (at your location), self-paced learning, online labs, and targeted skills assessments. Web of Security is an Authorized Training Center with EC-Council and is an approved cybersecurity training provider for the Department of Homeland Security. You can see some of the training programs Chad has developed at *https://webofsecurity.com/risk-management*.